A Grafting Approach to Neural Network Construction

by

Abdallah Mousa Rezq Rayhan

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

June, 1996

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

A Grafting Approach to Neural Network Construction

BY

Abdallah Mousa Rezq Rayhan

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Science

June 1996

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by

Abdallah Mousa Rezq Rayhan

under the direction of his thesis advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee:

Dr. Hussein Almuallim (Chairman)

Dr. Sulaiman Al–Bassam (Member)

Dr. Rafik Braham (Member)

Department Chairman

Dean, College of Graduate Studies

19.6.96

Date

إهداء

الحمد لله رب العالمين الذي ابتدأ وحيه بـ (إقرأ باسم ربك الذي خلق ، خلق الإنسان من علق إقرأ وربك الأكرم ، الذي علم بالقلم ، علم الإنسان ما لم يعلم) ، وثنى تنزيله بـ (ن والقلم وما يسطرون). وافتتح كتابه العزيز بـ (الحمد لله رب العالمين) وهي السبع المثاني والقرآن العظيم. والصلاة والسلام على محمد المبعوث فينا رحمة للعالمين بشيرا ونذيرا وداعياً إلى الله بإذنه وسراجا منيرا ، معلم الإنسانية الخير ومخرجها من الظلمات إلى النور بإذن ربها ، فعليه خاتم الأنبياء والمرسلين أفضل صلاة وأزكى تسليم ما ذكره الذاكرون وغفل عنه الغافلون وعلى جميع رسل الله وأنبيائه الكرام الطيبين.

أما بعد : فهذه باكورة إنتاجي العلمي، أهديها على ما فيها ومالها وما عليها، ثمرة يانعة بسعادة غامرة وقلب عامر بالوفاء والحب إلى من كان السبب في إيجادي بعد الله ووجودي، ومن حببه إلي العلم وزينه في قلبي فرغبني فيه، وحضني عليه، وإلى معلمي الخير والباذلين فيه كل غال ونفيس، أينما كانوا وحيثما وجدوا، وإلى الموجدين لهذا الصرح العلمي الواعد بكل خير والذي أشرفه بالانتساب إليه والنهل من معينه الذي لا ينضب وجميع القائمين عليه وإلى كل الذين منحوني النصح ومدوا لي يد العون بسخاء فذللوا لي جميع الصعاب وأخذوا بيدي إلى النجاح والفلاح. إلى هؤلاء وهؤلاء أقدم هذا العمل المتواضع على ما بذلت فيه من عصارة فكر وجهد مضن، وما أمضيته فيه من سويعات عمري وشبابي أحتسبها رخيصة في سبيل العلم النافع على مدى الدهور والأعوام ، وراجيا لكم أيها الساحة الكرام ولهم على الدوام الأمن والعافية وحسن المثوبة في الدنيا والآخرة. وكما بدأت هذا الإهداء أختمه بالحمد لله رب العالمين والصلاة والسلام على رسول الله.

والسلام عليكم ورحمة الله وبركاته

عبدالله موسى رزق ريحان

Acknowledgments

All praise be to Allah for his limitless help and guidance. Peace and blessings of Allah be upon his prophet Muhammad.

Acknowledgment is due to King Fahd University of Petroleum and Minerals for the generous help and support for this research.

I would like to express my profound gratitude and appreciation to my thesis chairman, Dr. Hussein Almuallim, for his guidance and patience throughout this thesis. His continuous support and encouragement can never be forgotten. I would also like to thank Dr. Rafik Braham, and Dr. Sulaiman Al-Bassam, for their consistent support and valuable suggestions and comments.

I also wish to thank the faculty, and the staff members of the Information and Computer Science Department for their support. Special acknowledgment is due to my friends Jammil Qawwas, Mohammad Al-Mughrabi, Tareq Al-Mughrabi, Mutlaq Al-Mutlaq, Maher Abu-Mutlaq, and Maher Al-Sharif for their encouragement and good wishes. Finally, special thanks must be given to my family for their encouragement and moral support.

Contents

List of Tables

List of Figures

Abstract

Name: Abdallah Mousa Rezq Rayhan

Title: A Grafting Approach to Neural Network Construction

Major Field: Computer Science

Date of Degree: June, 1996

The size of a neural network is mostly determined by trial and error and influenced by the designer's experience and the anticipated complexity of the problem. The rising demand for neural network applications necessitates better structured techniques to build neural networks than the existing ones. A review the back-propagation training algorithm and the main problems and limitations of learning is given in this thesis. The thesis also includes a survey of the main techniques reported in the literature, which dynamically construct or prune feedforward networks. A new approach to construct feedforward neural networks is then proposed. This approach is based on grafting the desired network from several already trained networks. We present experimental results on artificial and natural domains comparing the proposed grafting algorithm to networks with predetermined size, dynamic node creation, and node pruning. These results suggest that the proposed grafting algorithm achieves superior performance over other techniques in most of the tested domains.

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

June, 1996

بسم الله الرحمن الرحيم

خلاصة الرسالة

الإسم : عبد الله موسى رزق ريحان

عنوان الرسالة : تكوين الشبكات العصبية بطريقة التطعيم

التخصص : علوم الحاسب الآلي

تاريخ الشهادة : محرم ١٤١٧ هـ

يتم تحديد التركيب البنائي للشبكات العصبية عموما عن طريق التجربة والخطأ ، ويتأثر بخبرة المصمم ودرجة تعقيد المسألة المطلوب حلها. ولقد أصبحت الشبكات العصبية تتطلب طرقا أكثر تنظيما لبنائها نظرا لأهمية تطبيقاتها . هذه الدراسة تعرض طريقة تصحيح الخطأ الإسترجاعي لتدريب الشبكات العصبية وما يترتب عليها من صعوبات. كما تعرض أهم الطرق التقليدية لبناء الشبكات . ومن ثم تقترح هذه الدراسة طريقة جديدة لبناء الشبكات العصبية بالتطعيم من شبكات أخرى مدربة مسبقا. وتعرض الدراسة كذلك مقارنة نتائج طريقة التطعيم الجديدة لحل بعض المسائل المصطنعة والطبيعية مع نتائج الطرق التالية : الشبكات العصبية ذات الحجم المحدد مسبقا ، وتكوين الخلايا ديناميكيا ، وتشذيب الخلايا العصبية. وتخلص هذه الدراسة إلى أن طريقة التطعيم تتفوق في أغلب الأحيان على الطرق الأخرى .

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران – المملكة العربية السعودية

محرم ١٤١٧ هـ

Chapter 1

Introduction

Neural networks have been intensively studied in the last decade. They are now being used with promising success as classification models in a variety of applications. However, full understanding of how neural networks capture and process information has not been achieved. A neural network is trained by subjecting it to stimuli with (or without) correct responses (training examples) and adjusting its internal state by tuning the weights to minimize the inconsistency in classifying the training examples. Neural network training involves a trade-off between generalization and overfitting [31]. The generalization performance measures how well a network can generalize from the training examples to the underlying function. In other words, it measures how well a network can perform on the unseen examples. The *overfitting* problem occurs when the network memorizes the training examples very well but fails to recognize the testing examples well.

It has been found that the performance of neural networks is sensitive to learning parameters, initial conditions and the possibility of local minimum trapping. In particular, it is hard to determine the right size of the network that would lead to the best performance.

Very large networks tend to overfit the training examples and generalize poorly for the testing examples. On the other hand, too small networks are not capable of either memorization nor generalization. A simple rule to achieve good generalization is to use the smallest network which will fit the training examples [22]. The problem, however, is that there is no known method to determine the smallest size of the network that will learn a given set of training examples.

A recent approach to overcome the overfitting problem is to train a large network. Pruning heuristics are then applied to minimize its complexity by removing those neurons and/or connections which have the least effect on the network performance. By training a network with large initial size, it is more likely to learn quickly with less sensitivity to initial conditions; and pruning it would reduce its complexity and enhance its generalization performance [22].

Another approach called dynamic construction works in the reverse direction. It starts by a very small network and uses incremental learning by adding more hidden nodes whenever the network is trapped in a local minimum. Both of these approaches are promising but they do not take advantage of the history of learning. In this new grafting approach introduced in this thesis, the history of learning

plays an important role. The objective of our approach is to graft a network from several networks by selecting the parts which have useful knowledge of the training examples.

This thesis is organized as follows. In Chapter 2, we will review the back-propagation algorithm. In Chapter 3, we will review the pruning and dynamic constructions reported in the literature. In Chapter 4, we will present our new grafting algorithm to construct neural networks. Chapter 5 will include simulations and comparisons of our grafting algorithm with other related methods. In Chapter 6, we will present the conclusion of the thesis and future work.

Chapter 2

Learning in Back-Propagation Neural Networks

2.1 Introduction

The learning process in humans starts in the early time of life and continues afterwards. Modeling such kind of learning, however, is a very challenging task. Recognizing the surrounding environment is one of the early tasks a child tries to master. Several artificial approaches have evolved for modeling the recognition process such as the statistical, syntactic and neural approaches. The statistical and syntactic approaches, for which the underlying process is quite well understood, are used in several applications of pattern recognition with some success. The neural approach, for which we do not have yet a satisfactory level of understanding, have achieved

similar success in wide areas of applications.

Learning in neural networks can be viewed as black box training when compared to statistical or syntactic approaches in pattern recognition. The neural networks are sometimes referred to as connectionist models or massively parallel processing models. Basically, a neural network is a network of simple processing elements called nodes, neurons, neural cells or processing units. Neural networks are categorized by the way they are interconnected (topology), the node characteristics and/or the training (learning) algorithm being used.

Without loss of generality, neural network topologies can be divided into three groups: complete graphs, bipartite graphs and incomplete k-bipartite graphs. The complete graph network, Figure 2.1(a), is known as the content addressable memory model or sometimes called Hopfield model. The bipartite graph model network, Figure 2.1(b), is known as the self-organizing maps or the Kohonen network. The k-bipartite graph network, Figure 2.1(c), is what is generally known as feedforward network.

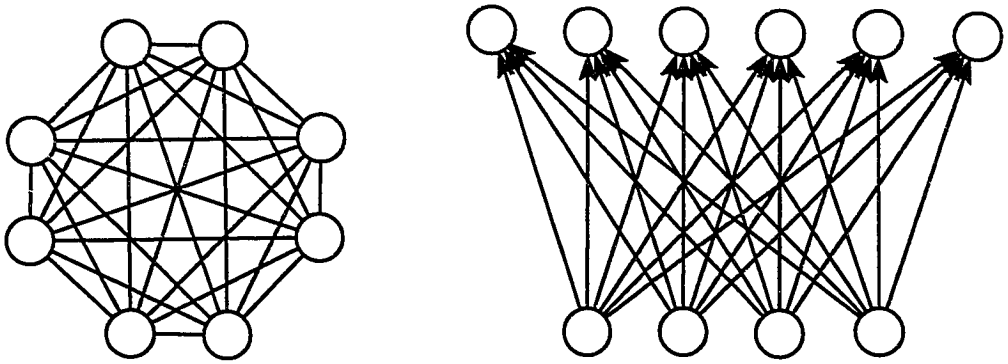
A neural node, basically, applies a function on the weighted sum of the input vectors plus a bias value, see Figure 2.2. The node can be modeled as a linear or nonlinear function as shown in Figure 2.3. A node fires a value which represents the *activity* of the node. In the linear case, the activation value of $f(net)$ is 1 if net is above a certain threshold θ, or 0 if net is below θ as illustrated in Figure 2.3(a). In the nonlinear case, the activation value is a continuous function between 0 and 1.

Several continuous differentiable functions can be used such as the *sigmoid* or *tanh* functions . The most common sigmoidal function is the logistic function shown in Figure 2.3(b),

$$f(net) = \frac{1}{1 + e^{-net}} . \tag{2.1}$$

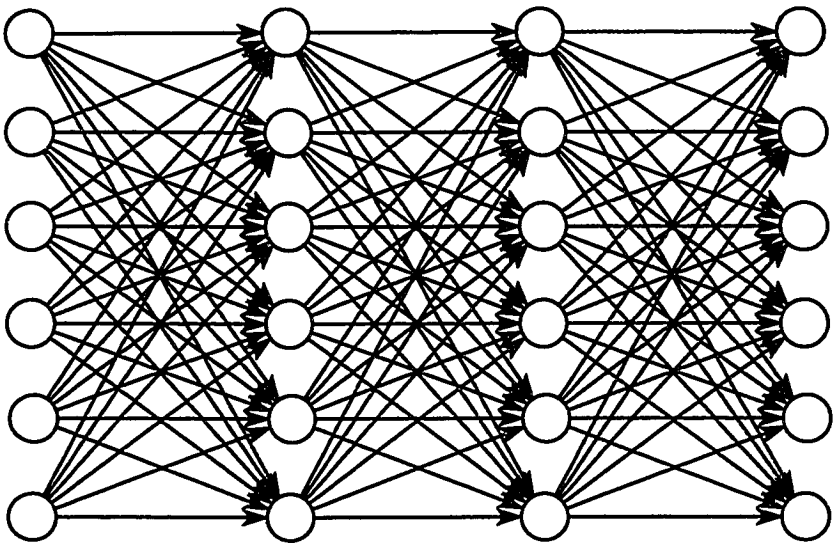
Learning can be attributed as supervised or unsupervised. In unsupervised learning, a network is subjected to stimuli (input vectors) with no correct responses (output vectors). Kohonen algorithm is an example of unsupervised learning for the self-organizing maps. In supervised learning, on the other hand, the network has the advantage of being taught the responses along with the stimuli. In the Hopfield model and feedforward networks, supervised learning is applied [16].

In the supervised learning networks, the patterns can have static or dynamic nature yielding two types of networks, static and dynamic. When patterns are static, they are time independent such as recognizing pictures or characters. The patterns are called dynamic when they are time dependent such as detecting movement in scenes or weather forecasting. The static networks are memoryless and do not preserve the state of processing. The backpropagation training algorithm for feedforward neural networks is an example of the static networks. The static networks can be used in various kinds of applications such as implementing logic functions [23], pattern recognition [8] and function approximation [18]. The dynamic networks have memory and preserve the state of processing, for example recurrent networks and Hopfield models. They are used in modeling real world systems which have non-



(a) Hopfield Model

(b) Kohonen Network



(c) Feedforword Network

Figure 2.1: The neural network topology for (a) Hopfield model, (b) Kohonen model, and (c) feedforward model

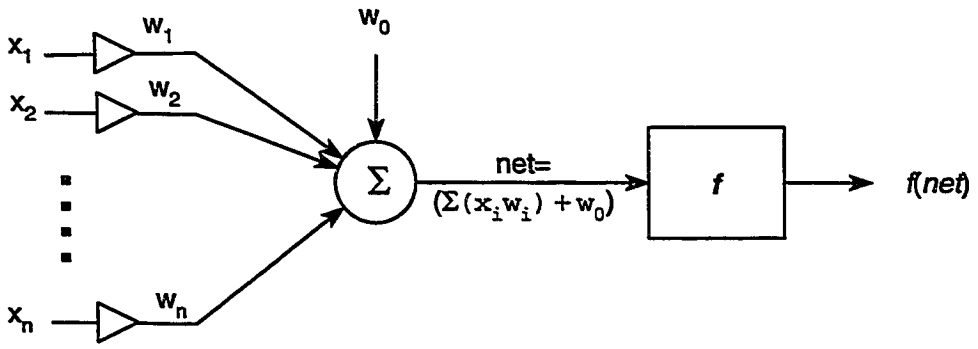
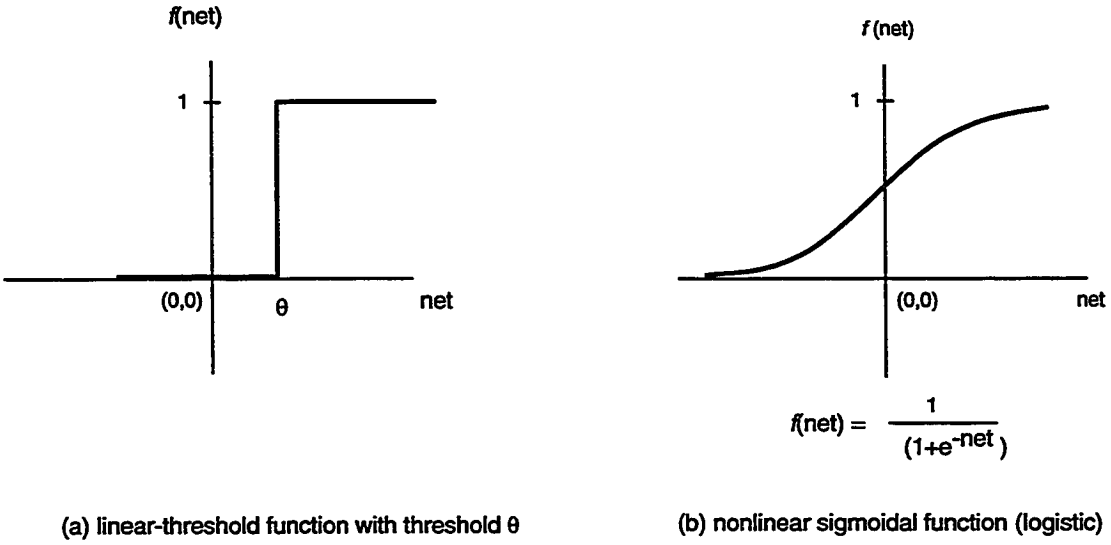


Figure 2.2: The neural node architecture



(a) linear-threshold function with threshold θ

(b) nonlinear sigmoidal function (logistic)

Figure 2.3: The node output functions for (a) linear-threshold function and (b) nonlinear sigmoidal (logistic) function.

linear nature such as rockets and spacecrafts control and finite state machines. In this thesis, we will be concerned only with supervised learning for feedforward neural networks. Therefore, the term learning will simply be used to mean supervised learning only.

Feedforward neural networks are typically used in supervised learning. A feedforward network consists of an input layer, an output layer, and one or more hidden layers. Each layer consists of one or more neurons. The layers are cascaded one after another starting with the input layer and ending with the output layer where in-between layers are interconnected by weights. A three-layer network is described as I-J-T, where I is the number of nodes at the input layer, J is the number of nodes at the hidden layer, and T is the number of nodes at the output layer. A network with more layers is described as I-J_1-$\cdots$-J_l-T, where J_1 is the number of nodes at the 1st hidden layer, and J_l is the number of nodes at the lth hidden layer. The set of neurons in a layer are connected to the neurons in the next higher layer. The connections between neurons are called weights. Each neuron would apply a function, linear or nonlinear, to the incoming weights, see Figure 2.3.

Several training algorithms have been developed to train feedforward neural networks. The backpropagation (BP) training algorithm which conducts gradient descent search to minimize the error with respect to a given set of training examples was developed by Rumelhart et al. in 1986 [23] and others [32, 21]. Since then it became a standard technique to train feedforward networks. Other techniques were

developed to enhance the performance of BP such as the conjugate gradient method, PROP and quickprop etc. [25]. In Section 2.2, we will review the BP algorithm for feedforward neural networks. In Section 2.3, we will discuss a few issues about learning in backpropagation neural networks (BPNN).

2.2 Training in Feedforward Neural Networks

Training a neural network involves teaching it a set of examples. The set of training examples are defined as ordered pairs of vectors,

$$\mathcal{S} = \{< \bar{i}_p, \bar{t}_p > | \, 1 \leq p \leq P\}$$

where P is the number of training examples, $\bar{i}_p$ is the pth input vector, and $\bar{t}_p$ is the pth target output vector. The input and target vectors are defined as,

$$\bar{i}_p =< i_{p1}, \ldots, i_{pi}, \ldots, i_{pI} > \quad \text{and} \quad \bar{t}_p =< t_{p1}, \ldots, t_{pt}, \ldots, t_{pT} >$$

where I is the number of neurons at the input layer, i_{pi} is the value of ith feature of the input vector $\bar{i}_p$, t_{pt} is the tth class of the target vector $\bar{t}_p$, and T is the number of neurons at the output layer. A network is trained when its weights are adjusted properly such that the error difference between the target vectors and the actual output vectors is minimal. The network attains a satisfactory level of consistency when our objective error function is minimized. Only then, it could be used as a classification model for a specific type of application.

For example, a network may be used to help in diagnosing diseases based on some given symptoms. Such a network would be trained to make a smart guess of a disease by feeding it with some relevant symptoms. Then, we may say that the network is trained properly when it is able to guess smartly the disease of some patient having certain symptoms.

The most common learning algorithm for training feedforward networks is the back-propagation algorithm that uses the generalized delta rule to adjust the weights to the proper values [23]. It conducts gradient descent search to minimize the error between the target and the output vectors. The total error is the sum of the error for all patter vectors,

$$E = \sum_{p=1}^{P} E_p. \tag{2.2}$$

The error function which we seek to minimize is,

$$E_p = \frac{1}{2} \sum_{k=1}^{T} (t_{pk} - o_{pk})^2, \tag{2.3}$$

where E_p is the error of the pth training example, T is the number of output nodes and t_{pk} and o_{pk} are the kth element of the target and the output vectors, respectively. To minimize the error, we need to minimize the difference between t_{pk} and o_{pk}. The value of o_{pk} is decreased or increased by changing the incoming weights to output node o_k since the neural cell performs a weights sum on its inputs. The node o_k is connected by J weights to the nodes in the lower layer. The weight between output node k and node j in the lower layer is w_{kj}. To decrease the value of E_p, we need

to quantify the incremental change of E_p due to the incremental change of weight w_{kj} as

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial w_{kj}}, \tag{2.4}$$

where

$$net_{pk} = \sum_{h=1}^{J} w_{kh} o_{ph}. \tag{2.5}$$

Using the chain rule to compute $\partial E_p / \partial net_{pk}$,

$$\frac{\partial E_p}{\partial net_{pk}} = \frac{\partial E_p}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial net_{pk}}. \tag{2.6}$$

For the kth output node, we can find $\partial E_p / \partial o_{pk}$ by differentiating Eq(2.3), so

$$\frac{\partial E_p}{\partial o_{pk}} = -(t_{pk} - o_{pk}). \tag{2.7}$$

Later, we will show how to calculate $\partial E_p / \partial o_{pk}$ for hidden node j. The function that the neural cell performs is assumed to be nondecreasing and differentiable. In our case, we will assume the logistic function,

$$f(net) = \frac{1}{1 + e^{-net}}. \tag{2.8}$$

The output of node o_{pk} is

$$\begin{aligned} o_{pk} &= f_k(\sum_{j=1}^{J} w_{kj} o_{pj}) \\ &= f_k(net_{pk}), \end{aligned}$$

and

$$\frac{\partial o_{pk}}{\partial net_{pk}} = f_k'(net_{pk}). \tag{2.9}$$

Rewriting Eq(2.6), we get

$$\frac{\partial E_p}{\partial net_{pk}} = -(t_{pk} - o_{pk}) f_k'(net_{pk}). \tag{2.10}$$

Since we want to minimize E_p, then we should move in the opposite direction,

$$\delta_{pk} = -\frac{\partial E_p}{\partial net_{pk}} = (t_{pk} - o_{pk}) f_k'(net_{pk}). \tag{2.11}$$

Using Eq(2.5) , $\partial net_{pj}/\partial w_{ji}$ is nothing but,

$$\frac{\partial net_{pk}}{\partial w_{kj}} = o_{pk}. \tag{2.12}$$

Therefore, by the definition of Eq(2.4) and using Eqs(2.11) and (2.12), we can write the change in weight as,

$$\Delta_p w_{kj} = \eta \delta_{pk} o_{pk}, \tag{2.13}$$

where η is the learning rate. In Eq(2.13) we have shown how to calculate the δ_{pk} for the output nodes but we still need to compute δ_{pj} for a hidden node j. Let us use Eq(2.5) to derive $\partial E_p/\partial net_{pj}$ for the hidden node j,

$$\frac{\partial E_p}{\partial net_{pj}} = \sum_{k=1}^{T} \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}}, \tag{2.14}$$

where

$$\frac{\partial net_{pk}}{\partial o_{pj}} = w_{kj}. \tag{2.15}$$

Recalling Eq(2.11), $\delta_{pk} = -\partial E_p/\partial net_{pk}$, to rewrite Eq(2.14) as

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} f_j'(net_{pj}) \tag{2.16}$$

$$= -f_j'(net_{pj}) \sum_{k=1}^{T} \delta_{pk} w_{kj}. \tag{2.17}$$

1. **Initialize** with random weights.
2. **While** the error is below a threshold ϵ **Repeat**

2.1 **Forward** examples through the network.

2.2 **Use** Eq(2.11) to calculate the δ's for output nodes and Eq(2.17) for the hidden nodes.

2.3 **Use** Eq(2.18) to update the weights.

3. Report the network.

Figure 2.4: Outline of the backpropagation algorithm

In order to prevent oscillations while searching for a global minima and to help escaping a local minima during training, a momentum term may be added to Eq (2.13) which will make the learning less sensitive to local changes,

$$\Delta_p w_{kj}(n+1) = \eta \delta_{pk} o_{pk} + \xi \Delta_p w_{kj}(n), \tag{2.18}$$

where ξ is the momentum and n is the iteration index.

The outline of the BP algorithm is shown in Figure 2.4. Initially, the network is initialized with random weights. Then, the examples are forwarded through the network. After each iteration, Eq(2.11) is used to compute δ's for the output nodes and Eq(2.17) for the hidden nodes. Then, Eq(2.18) is used to update the weights. The error is calculated after updating the weights to compute the new error. This

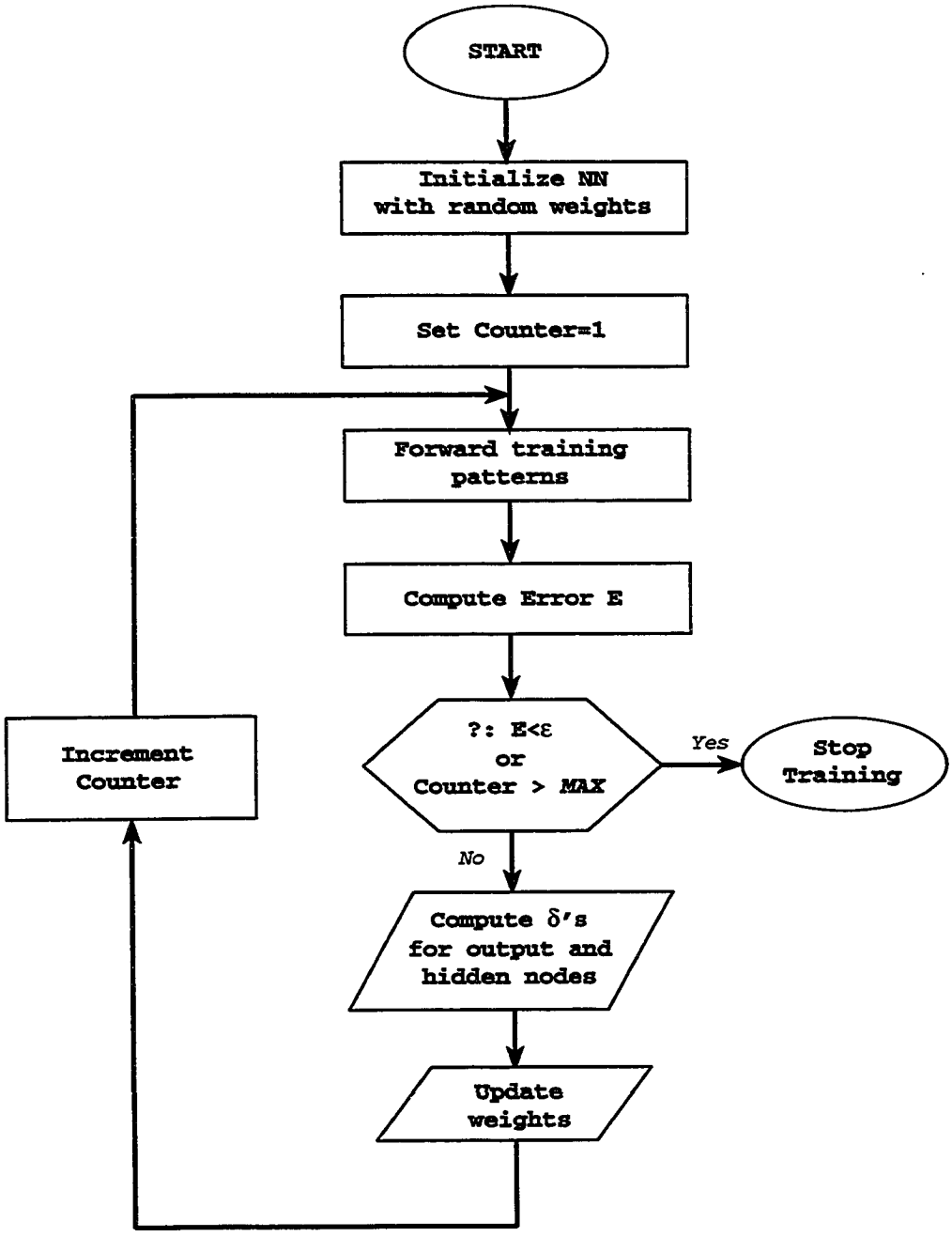


Figure 2.5: Flowchart of the backpropagation training algorithm

loop of error calculation and weights updating continues until the network produces minimal error or we decide to stop training after MAX number of iterations. Figure 2.5 shows the flowchart of the BP training algorithm. Further details can be found in Rumelhart et al. [23]. Throughout the thesis, we will assume a feedforward neural network with back-propagation training which uses gradient descent search to minimize the error.

The accuracy performance that a network $\mathcal{N}$ achieves is defined as the percentage of correctly classified classes over all examples. It can be formulated as

$$\text{Accuracy}(\mathcal{N}) = \frac{1}{P \times T} \sum_{p=1}^{P} \sum_{k=1}^{T} f(t_{pk}, o_{pk}), \tag{2.19}$$

where $f(t_{pk}, o_{pk})$ is defined as

$$f(t_{pk}, o_{pk}) = \begin{cases} 1 & \text{if } t_{pk} = 1 \text{ and } o_{pk} > 1 - \theta \\ 1 & \text{if } t_{pk} = 0 \text{ and } o_{pk} < \theta \\ 0 & \text{otherwise ,} \end{cases} \tag{2.20}$$

where θ is how far the actual output o_{pk} should be away from the target output t_{pk}, P is the number of examples, T is the number of classes in each example, t_{pk} is the target value of the kth class for the pth example, and o_{pk} is the output value of the kth class for the pth example.

2.3 Issues in BPNN Learning

Learning in BPNN involves several issues to be considered such as the time complexity of learning, conditions to achieve fitting of data and to achieve good generalization, necessary number of hidden layers and number of hidden nodes at each layer and the learning scalability to large problems. The fact that these issues are strongly dependant on each other makes it hard to discuss each issue separately.

Training a network involves adjusting weights to make the network consistent with the training examples. The training is known as the loading or fitting problem. It has been shown that the loading problem is NP-complete for a fixed size network [11]. Therefore, greedy heuristics which are based on gradient descent search such as BP algorithm have been developed to seek the desired training. The gradient descent search, however, does not always yield an exact mapping of the input vectors to the output vectors and is rather slow.

When to terminate the search is an important condition that affects the search greatly. Several termination conditions have been tried in the literature such as terminating when small change in the gradient occurs, achieving correct classification of the training examples, reaching a very small error value, controlling the number of iterations and monitoring the generalization of learning on the testing examples. The last condition may improve the generalization but is considered as computational intensive and inaccurate when the size of the training set is small or when the

available data does not represent the problem. The other conditions are parameter sensitive [10].

It has been observed in many applications that fitting the training data into a network may yield an unsatisfactory approximation of the underlying function. That is, fitting may lead to a network that, although memorizes the given examples perfectly, is not able to generalize well from those taught examples to unseen examples. The generalization performance is affected by three parameters which are the size of the training set, the complexity of the problem and the network size.

Usually, it is preferable to have a large set of training data because this would describe the underlying problem better. Unfortunately, there are no known practically applicable bounds on the size of the training data set that is sufficient to achieve a given level of generalization performance. Theoretically, bounds based on Vapnik-Chervonenkis dimension (VC-dim) which measures the capacity of the network have been reported in [2]. Although these bounds may help in deciding the sufficient size of the training set, they usually overestimate the size considerably.

Due to the black box nature of neural networks, there is not yet any measure to estimate the complexity of a given problem and hence, the optimal size of the network to use. Experimental studies have revealed an important phenomenon relating the size of a network to its performance. As the number of hidden nodes (degrees of freedom) increases, the network ability to memorize (fit the training examples) increases after training the network a fixed number of iterations and

its capability to classify unknown examples increases up to a certain point, see Figure 2.6. Afterwards, its ability to memorize continues to increase as the number of hidden nodes increases but its capability to generalize to the unseen examples decreases when the network is trained for the same fixed number of iterations [15, 22]. This suggests that there is a trade-off between fitting and generalization and the objective is to avoid overfitting, that is, to stop fitting at the right stage so the generalization performance is optimal. Two approaches that have shown some success towards this goal are the techniques of pruning and dynamic construction.

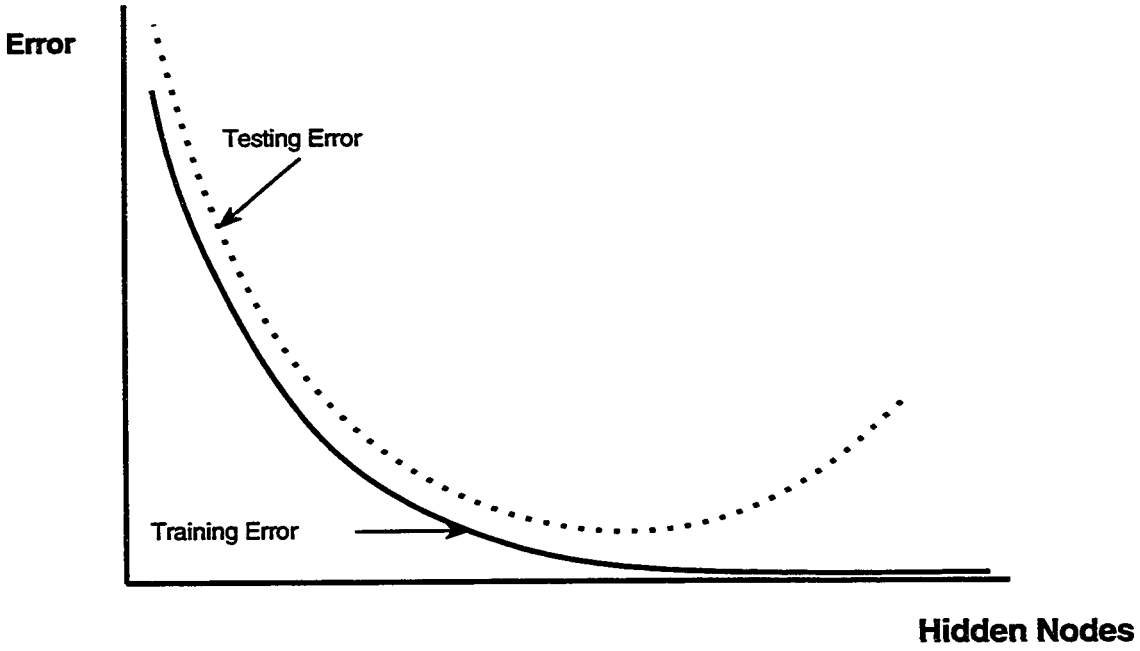


Figure 2.6: The effect of increasing the number of hidden nodes on the error of training and testing when the network is trained a fixed number of iterations.

In the pruning approach, the network is initially set to be much larger than nec-

essary. The network is then simplified by eliminating components, such as layers, nodes or connections, which affect the network the least. Such a method has been shown to improve generalization in certain problems. However, no criteria for deciding the minimum number of layers or minimum number of nodes have yet been reported.

In the other approach, dynamic construction, the network is allowed to grow in number of layers and hidden nodes till the given problem is solved hoping to improve the final generalization performance of the network. Thus, the network is initialized with a small number of hidden nodes. Then, more hidden nodes are added to the network as learning progresses. As the network grows, its ability to memorize and generalize develops till a satisfactory level of memorization and generalization is achieved. In the next chapter, we will discuss these two approaches in more detail.

Chapter 3

Pruning and Dynamic Construction Approaches

3.1 Introduction

Since the introduction of Rumelhart et al. paper [23] in 1986, several techniques have evolved to engineer the architecture of the back-propagation neural networks (BPNN). These techniques try to build a neural network that is capable of solving a given problem. The solution can be assessed by the ability of fitting the training examples in a network of small size and the capability of generalization on the testing examples. In Figure 2.6, we have seen the effect of the network size on fitting and generalization. This phenomenon is important, as we will see later, in developing techniques to determine the architecture of the network. The techniques

for determining the network architecture can be categorized into three approaches, ad hoc, pruning and dynamic construction.

The ad hoc approach is what has been widely used in most applications of neural networks [18, 5, 6, 8]. This design technique is based on trial and error. It is usually inspired by the experience of the researchers and the anticipated complexity of the problem. In most cases, a neural network goes under a rather lengthy and tedious process of designing the suitable architecture by conducting several simulations and adjusting various parameters. Results from previous research are sometimes helpful in this regard. For example, Rumelhart et al. [23] suggest that one hidden layer with n nodes are required to solve the n-bit parity problem. Although such suggestions may be applicable to a specific type of problems, such as the parity, no formula has been discovered yet to determine in advance the number of hidden nodes in general.

The role of nodes at the hidden layer(s) is to capture *abstract* features of the input vectors. When the neural network is properly trained and the number of hidden nodes is minimal, one could say that the hidden nodes have captured the important features of the training examples and thus the network could exhibit valid generalization on the testing examples. To achieve such a goal, first, a neural network with a large number of hidden nodes may be trained until it becomes as much possible consistent with the training examples. Since the number of hidden nodes is larger than necessary, some nodes would detect important features of the training data, while others would detect irrelevant features or be replicated copies of

others. By applying a careful *pruning* strategy, only those which capture important features will survive while the other nodes will be extracted.

On the other hand, by starting with a very small number of hidden nodes less features are captured. By applying a *dynamic construction* technique, which would increase the number of hidden nodes gradually, learning important features accumulates until convergence occurs.

These two approaches, pruning and dynamic construction, recently received wide interest by researchers in designing BPNN. In the rest of this chapter, our discussion will focus on the pruning and dynamic construction approaches and we will not discuss the ad hoc approach any further. In Section 3.2, we will discuss briefly the pruning and dynamic construction approaches reported in the literature. In Sections 3.3, 3.4 and 3.5, we will review the skeletonization, dynamic node creation and the quasi-Newton learning method, respectively.

3.2 Pruning & Dynamic Construction: An Overview

The goal in back-propagation learning is to find a global minimum of the cost function in the search space. The search space here is the space of all possible weight assignments, and the cost function measures *error*, or the remaining inconsistency with the training examples. The solution space is controlled by various parameters,

such as the number of layers, number of hidden nodes and learning parameters, e.g. learning rate, momentum and initial state. The learning algorithm should drive the neural network from the initial state to a final state at which the global minimum occurs. Such a learning algorithm should be least sensitive to the change in the learning parameters. However, the solution space is always affected by the change in the number of layers and number of hidden nodes.

Kolmogorov's theorem suggests that any continuous function can be mapped from the d-space, d input nodes, into the c-space, c output nodes, using exactly $(2d+1)$ nodes at the hidden layer in a three-layer neural network [16]. Therefore, the three-layer neural network is capable of implementing any continuous function. The question of interest to us here is "What is the minimum number of hidden nodes in a three-layer neural network needed to implement a particular function ?"

A simple approach to this question is to use the smallest possible network which will satisfactorily learn a given set of training examples. A minimal network that is able to fit the training examples is expected to generalize well on the testing examples. In the pruning approach, a large neural network is trained to fit the data, then a pruning technique is applied to repeatedly remove connections, hidden nodes or layers. Finally a small network is produced that achieves good fitting of the training data and expectedly exhibits good generalization performance of the testing data. The components which are removed from the large network are those which contribute the least to the error function to be minimized. By starting with

a large neural network, then pruning it, we overcome the problems of slow learning, sensitivity to initial conditions and local minima trapping, while achieving good fitting on the learning examples and good generalization performance on the testing examples [22].

For example, by pruning three hidden nodes, 3,5 and 1, from the network shown in Figure 3.1(a), then retraining the network after each pruning iteration, we will end up with a smaller network shown in Figure 3.1(d). This network is expected to perform at least as good as the unpruned one if not better. This approach is sometimes called pruning, trimming and elimination. We will use the term pruning to refer to this approach. In Section 3.2.1, we will briefly highlight the main pruning techniques reported in the literature.

On the other hand, in the dynamic construction approach, layers, hidden nodes and/or connections are added by some heuristic to build a small network which fits the training examples and gives good generalization on the testing data. Incrementally constructing a neural network in this fashion allows us to achieve high performance in terms of fitting, generalization and construction time.

For example, by adding three hidden nodes, 2,3 and 4, to the network shown in Figure 3.2(a), then training the network after each adding step, we will end up with a larger one shown in Figure 3.2(d). This later network is minimal in the number of hidden nodes and is expected to perform better than any of the previous versions. We will call this approach dynamic construction. In Section 3.2.2, we will briefly

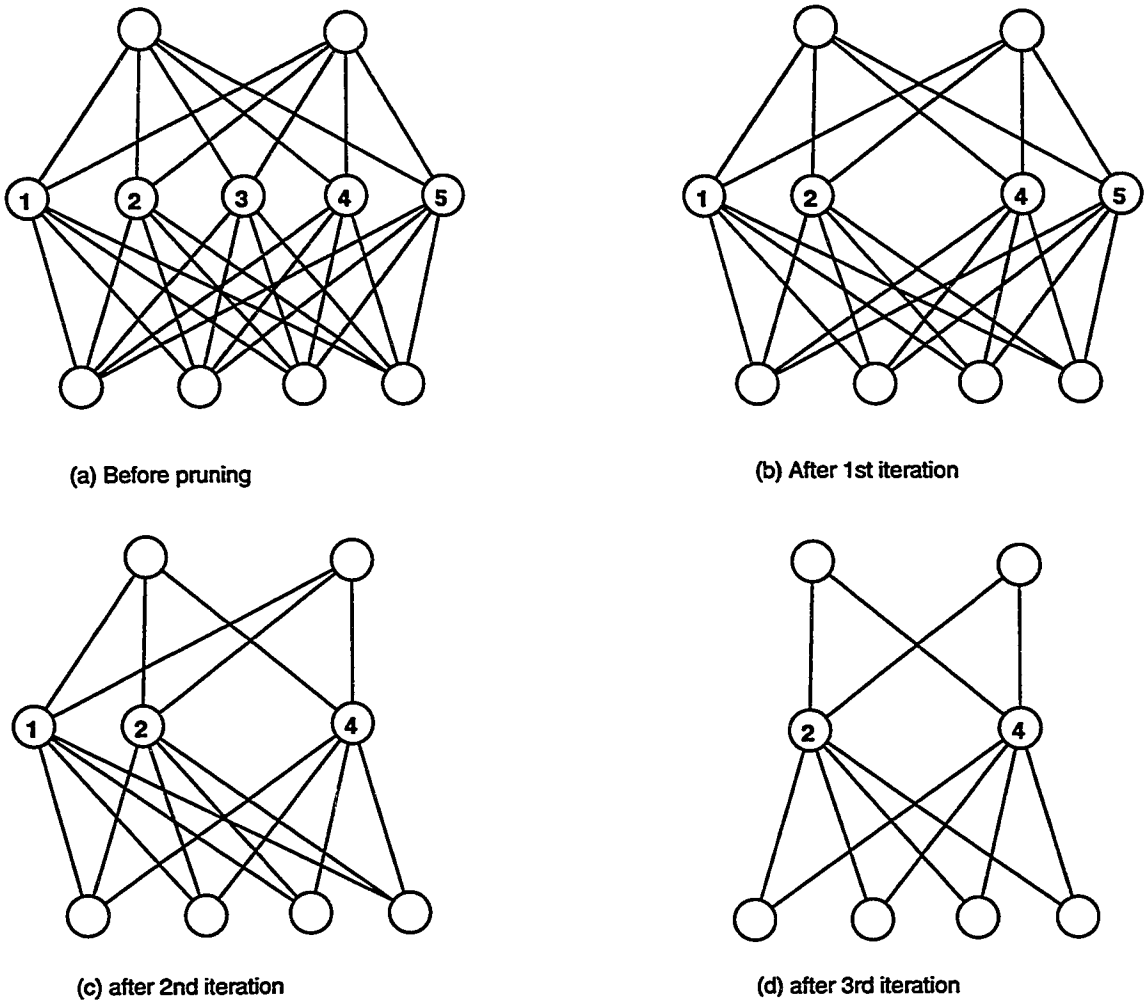


Figure 3.1: (a) The network before pruning. (b) The network after pruning node 3. (c) The network after pruning node 5. (d) The network after pruning node 1.

present the main construction techniques reported in the literature.

3.2.1 Pruning Approaches

Pruning algorithms developed so far generally fall into two main categories: connection (weight) pruning and node pruning. These two approaches follow the same pruning technique — they only differ in how to compute the pruning heuristic function and when to prune. The main pruning technique can be summarized in the general pruning framework shown in Figure 3.3. The effect of pruning hidden nodes on the error is illustrated in Figure 3.4. The question that pruning is trying to solve is "Starting with x hidden nodes, what is the minimum number of hidden nodes x' such that a maximum error of ϵ over the training data can still be achieved?"

One of the early attempts was the work of Sietsma and Dow in 1988 [27]. Less useful nodes are pruned based on how much they contribute to the network. Hidden nodes are pruned if their output values do not change over all input patterns. Also, when two nodes produce identical or inverse output values over all input patterns, one of them can be pruned. However, no general pruning algorithm was developed in [27]. Since then, several pruning algorithms have been developed which can be divided into two major groups: prune by sensitivity or prune by biased learning (penalty learning).

In pruning by sensitivity, the sensitivity of a node/connection is estimated. Then, the node/connection which contributes the least to the solution network is removed.

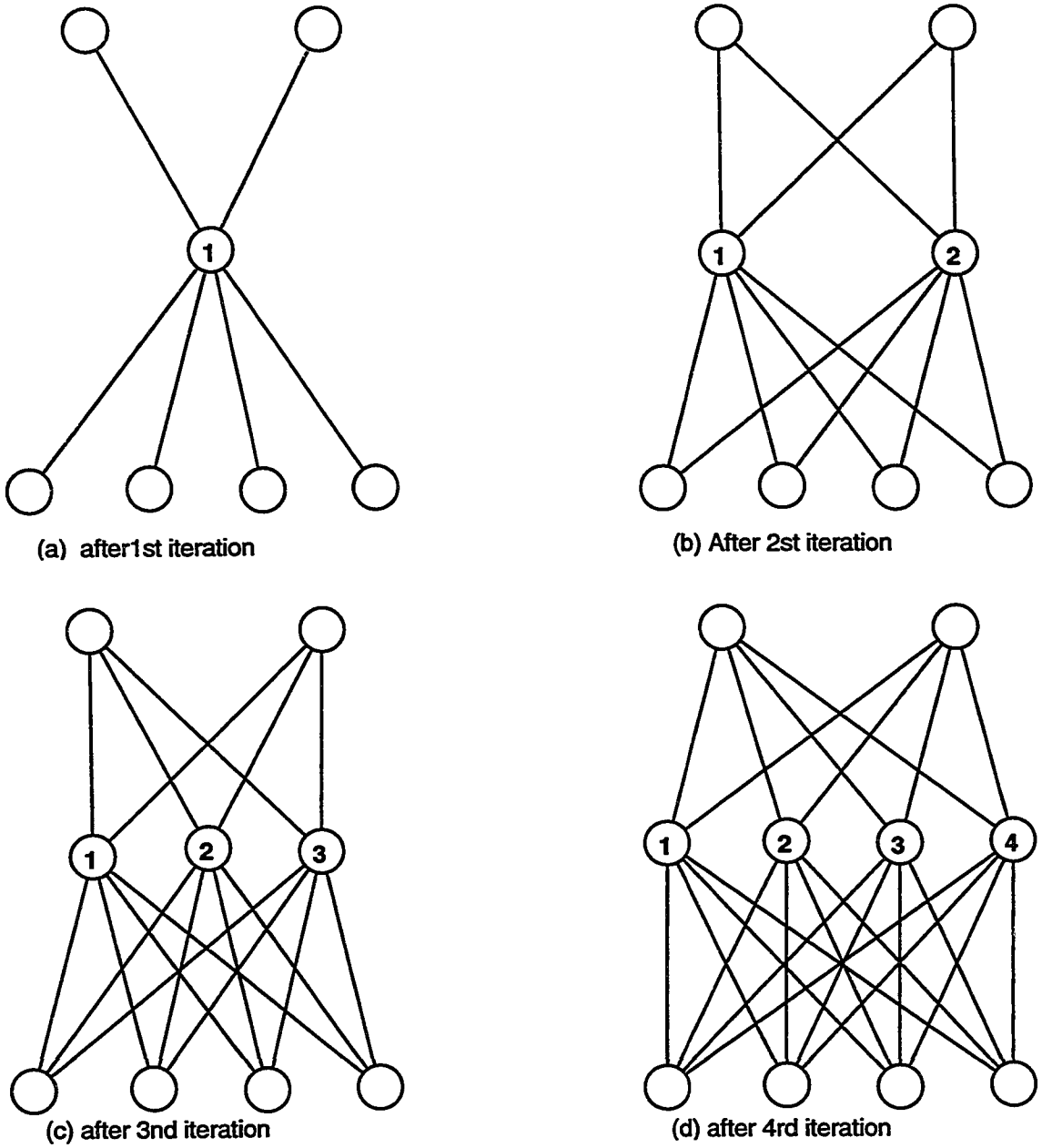


Figure 3.2: (a) The network before construction. (b) The network after adding node 2. (c) The network after adding node 3. (d) The network after adding node 4.

1. **Train** a large neural network over training examples until the error can not be made any smaller than ϵ or for **MAX** iterations.
2. **While** (error $< \epsilon$) and (#HiddenNodes $\geq$ Limit) **do**

2.1. **Store** network.

2.2. **Select** a node/connection based on algorithm α.

2.3. **Prune** the network.

2.4. **Retrain** the network after pruning to recover.

3. **Restore** last network.
4. Terminate.

Figure 3.3: Pruning framework of nodes or connections.

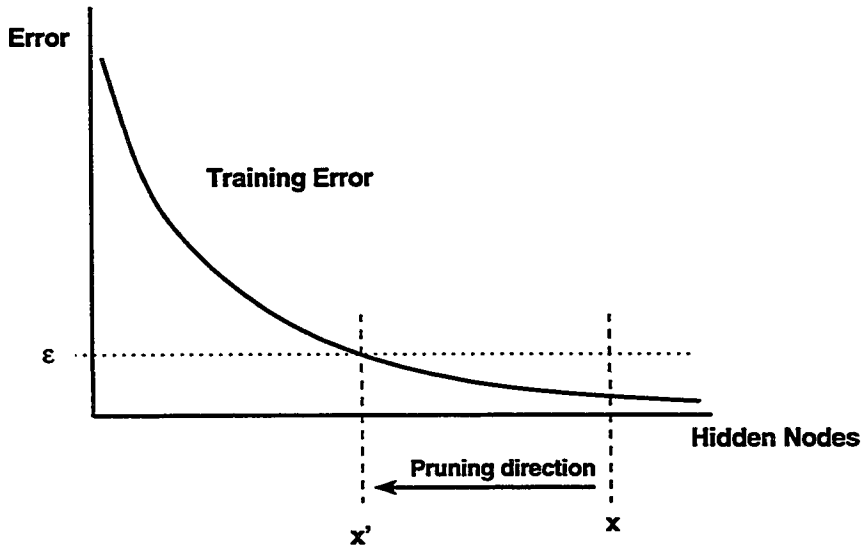


Figure 3.4: The effect of pruning on the error function.

The skeletonization technique developed by Mozer and Smolensky [19] estimates the sensitivity of a node by approximating the error obtained before pruning and the error introduced after pruning a particular node. In Section 3.3, the skeletonization method is reviewed in more detail.

Le Cun et al. [4] developed the Optimal Brain Damage (OBD) technique which is based on estimating the second derivatives of the error function with respect to connections by using the diagonal terms of the Hessian matrix to approximate the sensitivity of connections, and pruning the one with the least value. The OBD uses the back-propagation algorithm to train but uses Hessian matrix approximation to select which connection to prune next.

Karnin [13] simplified the skeletonization technique to prune weights rather than nodes. It is based on estimating the sensitivity of a connection by approximating the slope of the error obtained by the initial weight value and the error introduced by the final weight value. Burrascano [3] derived another sensitivity measure which relies on a probabilistic description of the training phase. It is similar to the one in [13] in being approximately proportional to the final weight value.

In an attempt to combine node pruning with connection pruning, Luk et al. [17] proposed a combined pruning algorithm. A node is pruned if its activity value decreases beyond a given activity threshold, and a connection is pruned if its sensitivity value is the lowest among all other connections and the degree of the node on either side of the connection is above a link threshold. The activity and link

thresholds are passed to the selection algorithm to decide which node/connection to select.

In the biased (penalty) learning, penalty terms are introduced to the cost function to be minimized. Such terms are to make learning biased towards constrained search in the search space. In other words, the network is penalized if it does not minimize some parameters. The common approach reported by Weigend et al. [31, 30] and Hanson and Pratt [9] is the weight decay. The biased learning allows large weights to persist while small weights to diminish. The large weights tend to be important because they contribute to the solution while small weights are less important because they contribute little to the solution. When weights decay, they become very small, almost zero, and can be eliminated safely with little perturbation of the solution network.

Kamimura [12] devised a technique which incorporates the entropy of hidden nodes activity in the update of connections. The entropy is to be minimized, then after training is performed the node with least sensitivity may be pruned.

Watanabe and Shimizu [29] introduced a new cost function to incorporate classification of binary patterns in which the node that causes least damage to the cost function is pruned.

Whitley and Bogart [33] devised a genetic algorithm to prune a large fully connected network. The state of a network is represented by a string (gene). It starts with one network and the population increases as connections are pruned yielding

new networks. The genetic pruning algorithm acts on a population of networks rather than just one.

3.2.2 Dynamic Construction Approaches

In the construction approach, rather than starting with a neural network larger than necessary, we start with a much smaller one. Then, the network grows slowly until the error over the training examples can be brought below a given threshold. The hidden nodes play the role of feature detectors. If the number of hidden nodes is less than necessary, some features of the training examples will not be learned, which would lead to poor generalization on the testing examples. Therefore, by driving a small network into a local minima, where the hidden nodes have learned to detect few features, then introducing more nodes to detect the features not being captured before, the network is given a chance to escape the current local minimum. Furthermore, it is enforced to capture more important features as it generalizes gradually through the training examples to more generalization on the testing examples.

The behavior of construction on the training error of the network as the number of hidden nodes increases is shown in Figure 3.5. Most of the construction techniques reported in the literature follow the same framework. A general construction framework is shown in Figure 3.6. Those techniques differ in what training algorithm they use, Steps 1 and 2.2, and how they construct the network, Step 2.1.

An early attempt to construct a neural network is the dynamic node construction

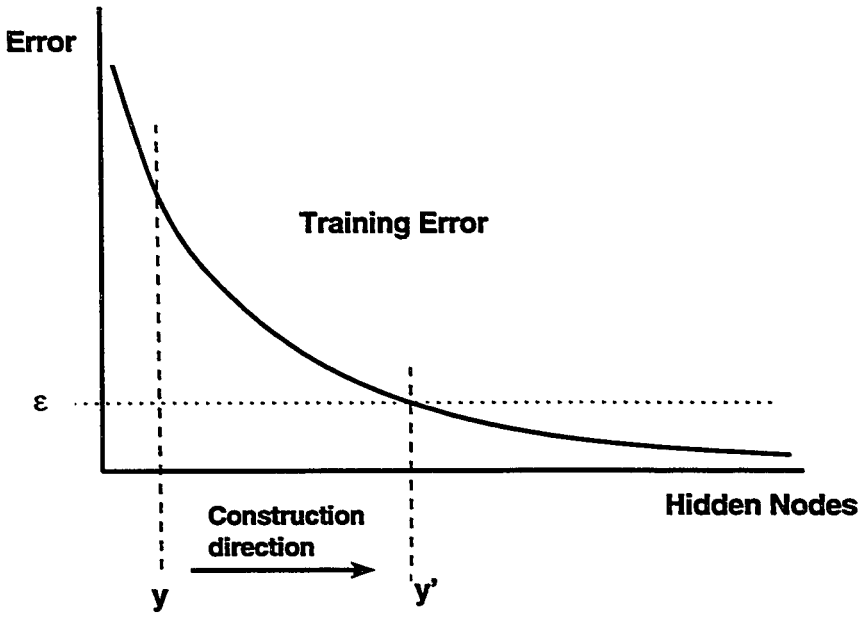


Figure 3.5: The effect of construction on the error function.

(DNC) [1]. The DNC method acts on a three layer network with variable number of hidden nodes. The hidden nodes are added as the network develops to learn a specific task. The added nodes are initialized with random weights. The added nodes are only connected to the output and input nodes. The back-propagation algorithm is used to train the network. Further details of this method are given in Section 3.4.

A variation of the DNC is the dynamic construction with SR1/BFGS learning [26] in which the architecture of the network is preserved. The technique starts with two hidden nodes and more nodes are added as the network evolves. The training algorithm is called SR1/BFGS and is based on the quasi-Newton method. This method will be reviewed in more detail in Section 3.5.

1. **Train** a small neural network over training examples until the error can not be made any smaller than ϵ or for **MAX** iterations.
2. **While** (error $>$ ϵ) and (#HiddenNodes $<$ Limit) **do**
2.1. **Change** the network by construction algorithm α.
2.2. **Retrain** the network after construction.
3. Terminate.

Figure 3.6: Construction framework of nodes or connections.

Fahlman and Lebieri [7] developed the Cascade-Correlation Learning (CCL) network. When a hidden node is added to CCL network, its incoming weights are connected to the outgoing connections of the input and all previous hidden nodes and its outgoing connections are connected to output nodes as depicted in Figure 3.7. Initially the network has no hidden nodes, then hidden nodes are added gradually to reduce the error. The quickprop [25] learning algorithm is used to train the network in two phases. In the first phase, incoming connections between the added node and previous nodes are modified by the quickprop algorithm. In the second phase, only outgoing connections from the hidden nodes to the output nodes are modified by the quickprop algorithm. Figure 3.7 illustrates the architecture of the CCL network.

Schiffmann and Werner [24] describe a genetic algorithm to construct neural networks. The networks are represented by genotypes and phenotypes which encode

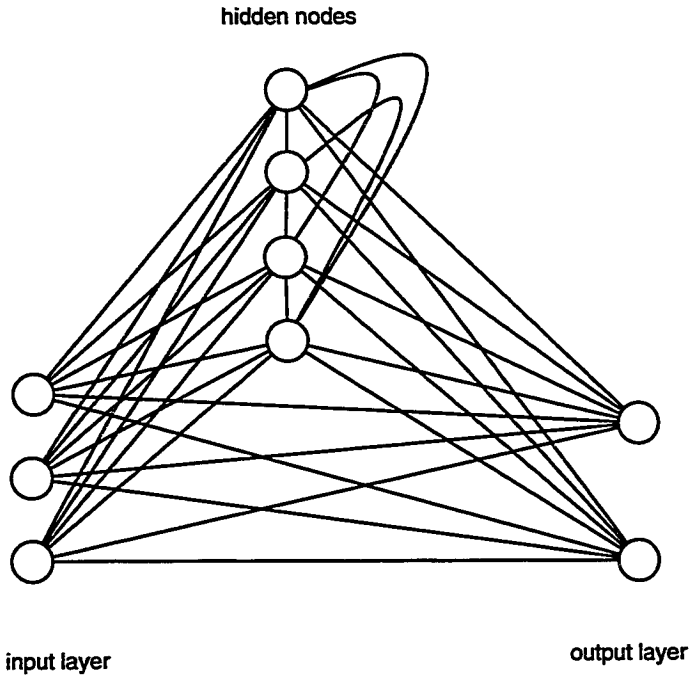


Figure 3.7: The architecture of the CCL network.

the architecture and the network, respectively. The genetic algorithm modifies the genotypes. The changes in the genotypes are reflected into the phenotypes, which are trained to solve a particular problem. This technique starts with random population of networks of different architectures. The genetic algorithm acts on the population to construct new networks and as more networks are genetically constructed they are added to the population.

3.3 Skeletonization Method

The skeletonization is a node pruning method based on estimating the sensitivity of hidden nodes of a neural network during training. It was developed by Mozer and Smolensky in 1989 [19]. It is based on repeatedly deleting the node whose removal contributes the least to the error function, E. A measure of relevance, ρ_i, for a node i is defined to be the error when the node is deleted minus the error before deletion,

$$\rho_i = E_{\text{without i}} - E_{\text{with i}} \cdot \tag{3.1}$$

Rather than computing ρ_i directly for every node in every iteration, Mozer and Smolensky approximated ρ_i by introducing α_i which gates the flow of activity for the weights leaving node i,

$$o_j = f(\sum_i w_{ji}\alpha_i o_i) \ , \tag{3.2}$$

where o_j is the activity of node j, w_{ji} is the weight from node i to node j, and f is a nonlinear function. When $\alpha_i = 1$, then the node behaves normally, but if $\alpha_i = 0$, the

node has no influence on the computation of the activity of node j. The relevance measure for node i is defined in terms of α_i as

$$\rho_i = E_{\alpha_i=0} - E_{\alpha_i=1} \ . \tag{3.3}$$

In order to approximate ρ_i, Mozer and Smolensky used the derivative of the error with respect to α_i, $\partial E/\partial \alpha_i$

$$\lim_{\gamma \to 1} \frac{E_{\alpha_i=\gamma} - E_{\alpha_i=1}}{\gamma - 1} = \left. \frac{\partial E}{\partial \alpha_i} \right|_{\alpha_i=1} \ . \tag{3.4}$$

Letting $\gamma = 0$,

$$\frac{E_{\alpha_i=0} - E_{\alpha_i=1}}{-1} \simeq \left. \frac{\partial E}{\partial \alpha_i} \right|_{\alpha_i=1} \ . \tag{3.5}$$

Therefore, the relevance ρ_i in Eq(3.3) is approximated by

$$\rho_i \simeq - \left. \frac{\partial E}{\partial \alpha_i} \right|_{\alpha_i=1} \ . \tag{3.6}$$

The quadratic sum of errors is used during training. Since a better approximation of error is needed for small error values, Mozer and Smolensky have used the linear sum of errors to measure the relevance.

$$E^{\ell} = \sum_j |t_{pj} - o_{pj}| \ . \tag{3.7}$$

Therefore, the approximate relevance measure of Mozer and Smolensky is eventually rewritten as

$$\hat{\rho}_i = - \left. \frac{\partial E^{\ell}}{\partial \alpha_i} \right|_{\alpha_i=1} \ . \tag{3.8}$$

The computation of the above derivative using back-propagation algorithm is similar to the weights update in a back-propagation algorithm. Also, the approximation assumes that $\alpha_i = 1$. Therefore, α_i is not a parameter to be implemented in the neural network; α is used for notational convenience to estimate relevance. It has been found that $\partial E/\partial \alpha$ fluctuates strongly in time. Therefore, to suppress the fluctuations, Mozer and Smolensky have used an exponentially decaying average of the derivatives,

$$\hat{\rho}_i(k+1) = .8\hat{\rho}_i(k) + .2\frac{\partial E}{\partial \alpha_i}\ , \tag{3.9}$$

where t is the tth pruning iteration.

The skeletonization procedure for pruning is summarized in Figure 3.8. First, a network I-J-T, where I is the number of input nodes and T is the number of output nodes, with large number of hidden nodes J is trained to solve the given problem. Then, the sensitivity of each node is computed using Eq 3.9. The node with the least sensitivity value is pruned and training continues for the network to recover. The loop of training and pruning is repeated several times until a satisfactory level of error does not exceed certain threshold or the number of hidden nodes reaches Limit.

1. **Train** a large neural network I-J-T using BP over training examples until the error can not be made any smaller than ϵ or **MAX** iterations.
2. **While** (error $< \epsilon$) and (#HiddenNodes $>$ Limit) **do**
   2.1. **Store** network.
   2.2. **Select** hidden node with least sensitivity using Eq 3.9.
   2.3. **Prune** the network.
   2.4. **Retrain** the network after pruning.
3. **Restore** last network.
4. Terminate.

Figure 3.8: Outline of the skeletonization procedure.

3.4 Dynamic Node Creation (DNC)

The DNC method was developed by Ash [1] to study the effect of adding hidden nodes on a feedforwad neural network with one hidden layer, I-1-T. The network starts with one hidden node and continues adding more nodes whenever the slope of average squared error (SASE) curve, when BP training is applied, drops below a certain threshold,Δ_T,

$$\frac{a_t - a_{t-w}}{a_{t_0}} < \Delta_T \, . \tag{3.10}$$

where a_t is the error at time t, a_{t-w} is the error before w iterations and a_{t_0} is the error at t_0. A new node can be added only when the current network has exceeded w iterations.

The training algorithm used for DNC is the back-propagation algorithm. The hidden and output nodes perform the sigmoidal function on their inputs. The construction procedure is outlined in Figure 3.9. Initially, the network consists of three layers with one node at the hidden layer, I-1-T where I is the number of input nodes, T is the number of output nodes and one hidden node. The BP algorithm is used to train the network and using Eq 3.10 to trigger the flatting of the ASE curve. Then, a new node is added with random weights and the network is trained again using the BP algorithm. This loop of training and adding is repeated several times until a satisfactory level of error is reached or the number of hidden nodes reaches Limit.

1. **Train** a I-1-T network using BP over training examples until the error can not be made any smaller or for **MAX** iterations.
2. **While** (error $> \epsilon$) and (#HiddenNodes $<$ Limit) **do**

2.1. **Change** from I-h-T to I-$(h+1)$-T.

2.2. **Retrain** the network using BP until SASE $< \Delta_T$, (Eq 3.10).

3. Terminate.

Figure 3.9: Outline of the DNC procedure.

3.5 Dynamic Construction by Quasi-Newton Learning

A variant of the DNC is the dynamic construction using quasi-Newton learning algorithm developed by Setiono and Hue [26]. The quasi-Newton method is based on optimizing the error function of the BP algorithm, E, for a three-layer network I-j-T, where I is the number of input nodes, T is the number of output nodes and j is the number of hidden nodes. The square error function can be written as a function of two variables, the number of hidden nodes, j, and the weights, w,

$$f(j,w) = \sum_{p=1}^{P}(\sigma(\sum_{h=1}^{j}\sigma(\bar{i}_p w_{h\bar{I}})w_{1h}) - t_{p1})^2 \ , \tag{3.11}$$

where P is the number of examples, j is the number of hidden nodes, $\bar{i}_p$ is an I-dimensional input vector, $w_{h\bar{I}}$ is an I-dimensional vector between the input nodes

and the hidden node h, w_{1h} is the weight value between hidden node h and the output node 1, and t_{p1} is the value of the output node 1 for the pth example. The activation function used at the hidden nodes and the output node is the sigmoid.

Setiono and Hue modified the quasi-Newton method called Broyden-Fletcher-Goldfarb-Shanno (BFGS) which approximates the inverse of the Hessian matrix (H) iteratively. Setiono and Hue use two updating schema to update H. Firstly, they use the SR1/BFGS method which produces symmetric rank one matrix to update H^{k+1},

$$H^{k+1} = H^k + \frac{\left(\delta^k - H^k\gamma^k\right)\left(\delta^k - H^k\gamma^k\right)^T}{\left(\delta^k - H^k\gamma^k\right)^T\gamma^k}\,, \tag{3.12}$$

where

$$\begin{aligned}\delta^k &= \omega^{k+1} - \omega^k\,,\\ \gamma^k &= \nabla f(\omega^{k+1}) - \nabla f(\omega^k)\,,\end{aligned}$$

and

$$\omega = (y, z)\,.$$

To maintain the positive definiteness of H^{k+1}, Eq 3.12 is used only when

$$\left(\delta^k - H^k\gamma^k\right)^T\gamma^k > 0\,. \tag{3.13}$$

Otherwise, the BFGS scheme which produces symmetric matrix of rank at most two to update H^{k+1} is used,

$$H^{k+1} = \left(I - \frac{\delta^k(\gamma^k)^T}{(\delta^k)^T\gamma^k}\right)H^k\left(I - \frac{\delta^k(\gamma^k)^T}{(\delta^k)^T\gamma^k}\right) + \frac{\delta^k(\delta^k)^T}{(\delta^k)^T\gamma^k}\,. \tag{3.14}$$

1. **Train** a network I-2-T using SR1/BFGS over training examples until Eq 3.15 is not satisfied or **MAX** iterations.
2. **While** (Eq 3.15 is not satisfied) and (#HiddenNodes < Limit) **do**

2.1 **Change** from I-j-T to I-$(j+1)$-T.
2.2 **Retrain** the network after construction using SR1/BFGS.

3. Terminate.

Figure 3.10: Outline of the SR1/BFGS algorithm

The termination condition considered for the SR1/BFGS is

$$\|\nabla f(\omega^k)\| \leq 10^{-6} \max\{1, \|\omega^k\|\} \ . \tag{3.15}$$

The outline of the SR1/BFGS algorithm is summarized in Figure 3.10. First, a network I-1-T, where I is the number of input nodes and T is the number of output nodes, with two hidden nodes is trained to solve the given problem. Then, one hidden node is added with random weights. The addition changes the network architecture from I-j-T to I-$(j+1)$-T. The network is trained to become more consistent with training examples after the addition. The loop of training and adding is repeated several times until a satisfactory level of error falls below threshold or the number of hidden nodes reaches Limit.

Chapter 4

Grafting Neural Networks:

A New Approach

4.1 Introduction

The pruning and dynamic construction approaches discussed in Chapter 3 attempt to find the minimum number of hidden nodes in a three-layer network which is sufficient to solve a given problem. In the pruning approach, an oversized network is trained to fit the training data and pruning heuristics are then applied to improve the generalization on the testing data. Conversely, in the dynamic construction approach, the network gradually grows from a smaller number of hidden nodes to a larger number by adding nodes and/or weights and forcing the hidden nodes to capture more new concepts and generalize better while fitting the training data.

A major disadvantage of these approaches is the possible loss of knowledge during training. Training a neural network involves carrying out several attempts before memorization takes place. When applied on a network, the pruning and dynamic construction techniques try to enhance the generalization performance in each attempt. If a satisfactory level of accuracy on the training and testing examples is achieved, training is considered successful and the network is reported. Otherwise, a new trial has to be carried out. The knowledge gained during the training process of unconverged networks is lost. This partial knowledge is valuable and may be used to construct a network with a satisfactory level of memorization and generalization performance.

Further, the pruning approaches have other disadvantages. Although an oversized neural network is able to easily fit the training data, such a network suffers from slow training, the possibility of being trapped in a local minima, as well as poor expected generalization performance.

In both the pruning and dynamic construction approaches, the network size is mainly determined by trial and error. When a pruning heuristic is applied to prune an oversized network, the benefits of previous trials are lost and not used to build a useful network. Most of the time is wasted in trying to teach the oversized network and its pruned versions and changing various learning parameters. As the oversized network is pruned further, it could easily be driven into a local minima.

The dynamic construction approaches have similar disadvantages. The training

algorithms are rather slow. The hidden nodes are added with random weights which could drive the network into a local minima. Also, they do not take any benefit from previously trained networks.

In the following section, we introduce a new approach to neural network construction in which the history of learning plays an important role. The key idea in our approach is to initially train a number of networks in parallel, and then construct the final network by combining those parts of the initial networks that appear to have captured relevant knowledge.

4.2 Description of the Grafting Algorithm

The role of the nodes at the hidden layer is to capture abstract features of the input vectors. Thinking optimistically, if the network is properly trained and if the number of nodes at the hidden layer is minimal or close to minimum, then we could say that these nodes have captured the important features of the training examples and thus could contribute to valid generalization on the testing examples. In reality, however, this is difficult to achieve. A normal training session would usually lead to hidden nodes capturing a mixture of relevant and irrelevant features. The basic idea in our approach is that by training several networks with different number of nodes at the hidden layers, the overall set of hidden nodes in these networks is likely to cover the important features of the training examples in addition to some irrelevant

features. Given the set of all hidden nodes in the trained networks, our goal is to select a minimal and sufficient subset of these nodes—that is, a collection of nodes that covers all and only the important features. These nodes are then grafted to build the targeted network.

Let us first describe the notations that we will use in this chapter. The architecture of a three layer network can be described as I-J-T, where I is the number of input nodes, J is the number of hidden nodes, and T is the number of output nodes. We will refer to the ith input node as x_i, jth hidden node as y_j and tth output node as z_t. The set of incoming weights to a hidden node y_j (those weights between the input layer and y_j) is denoted α_{y_j}. The set of incoming weights to an output node z_t (those weights between the hidden layer and z_t) is denoted β_{z_t}. The set of all hidden nodes is denoted $Y = \{y_1, \cdots, y_j, \cdots, y_J\}$. These notations are illustrated in Figure 4.1.

Our declared goal is to construct a network $\mathcal{N}$ that satisfactorily fits the training examples while minimizing the number of hidden nodes. We approach this goal by first training R networks independently each having I input nodes, T output nodes and some arbitrary number of hidden nodes. Denote by Y_r the set of hidden nodes in the rth network and let Γ to be the set of the hidden nodes; that is of all the R networks, $\Gamma = Y_1 \cup Y_2 \cup \cdots \cup Y_R$. From all the elements in Γ, suppose that a set $\gamma \subset \Gamma$ of hidden nodes was chosen using some strategy (as will be described later). Based on γ, denote by $\mathcal{N}(I, T, \gamma)$, for some integers I and T, a network constructed

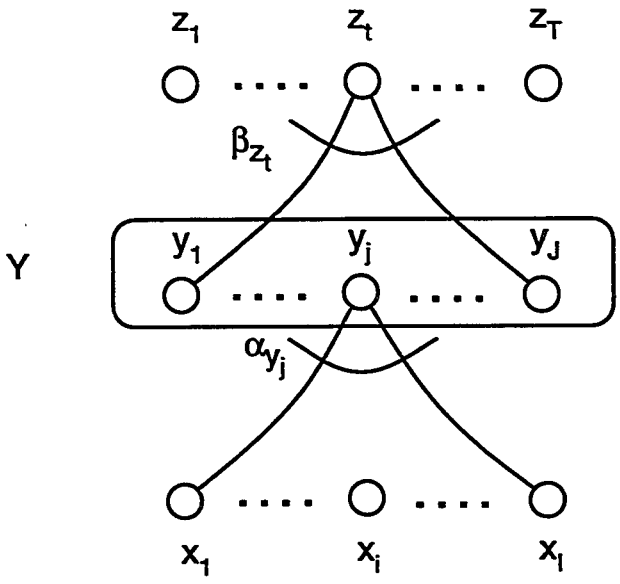


Figure 4.1: A three-layer network *I-J-T*.

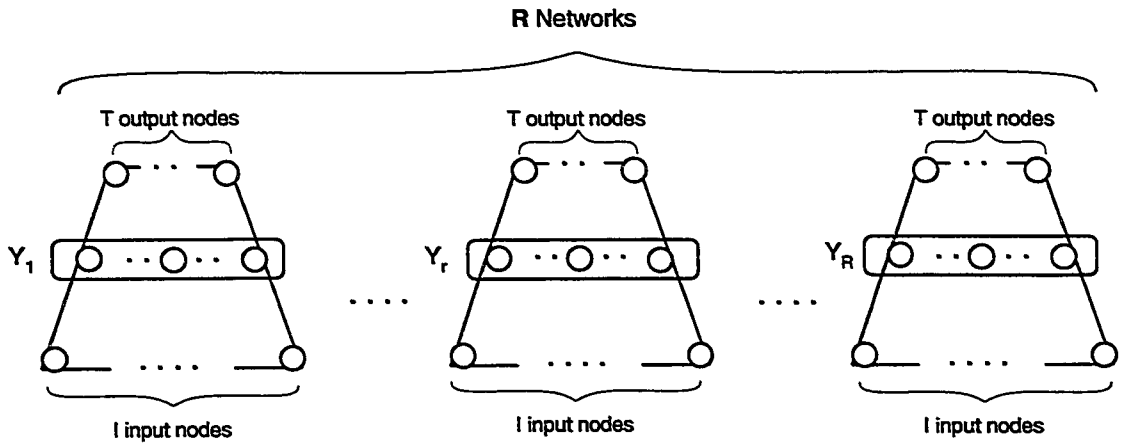


Figure 4.2: Description of the R initial networks.

as follows. $\mathcal{N}$ is an I-$|\gamma|$-T architecture. That is, $\mathcal{N}$ has I input, $|\gamma|$ hidden and T output nodes. The weights between the input and hidden layers are imported from the R networks. Namely, for each $y \in \gamma$, the incoming weights to y will just be α_y as determined in the R networks. The weights between the hidden and output layers are initialized by random values. The network $\mathcal{N}$ is then trained by updating the weights between the output and hidden nodes while freezing the weights between the hidden and input nodes until the network becomes as consistent as possible with the training examples. Note that in the above setting, the network $\mathcal{N}$ can be constructed once $\gamma \subset \Gamma$ is determined.

With the above definitions, our construction problem can now be formulated as follows: "Given R different trained networks (as those shown in Figure 4.2) and a set of examples $\mathcal{S}$, find a minimal subset $\gamma \subset \Gamma = \{Y_1 \cup Y_2 \cup \cdots \cup Y_r \cup \cdots \cup Y_R\}$, such that $\mathcal{N}(I, T, \gamma)$ is satisfactorily consistent with the training examples $\mathcal{S}$"

To achieve our goal, we will follow a greedy strategy to graft hidden nodes from Γ into $\mathcal{N}$. The idea of our grafting algorithm is to repeatedly select a hidden node y from Γ whose inclusion in γ maximizes the classification accuracy of $\mathcal{N}(I, T, \gamma)$. Having determined node y, it is then added into γ and removed from Γ. This process of *selecting*, *adding* and *removing* is repeated several times until the network $\mathcal{N}$ achieves a satisfactory level of classification accuracy with respect to the training examples.

The grafting algorithm is explained in the following example. Figure 4.3(a) depicts three initially trained networks using BP. Each network consists of 2 input nodes, 1 output node and 2 to 4 hidden nodes. Initially, Γ is the set of all hidden nodes, $\Gamma = \{y_1^1, y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_2^3, y_3^3, y_4^3\}$, where y_j^r is the jth hidden node in the rth network and γ is empty. The scores of the elements in Γ are then calculated. The score of each hidden node (HN) $y \in \Gamma$ is calculated as the classification accuracy of $\mathcal{N}(I, T, \gamma \cup \{y\})$. Let us assume that the results of these calculations is as shown in Figure 4.4. In the 1st iteration, the hidden node y_1^1 gave a score of 0.26. Therefore, y_1^1 is added to γ and removed from Γ. Having found y_1^1, the network $\mathcal{N}(I, T, \gamma)$ is then fine-tuned by running a few training iteration using BP and subjecting all weights to change in order to improve its classification accuracy. As given in Figure 4.4, we assume that the classification accuracy of 0.5 was achieved. Figure 4.3(b) shows $\mathcal{N}$ after the end of the 1st iteration. In the 2nd iteration, hidden node y_2^3 gave the maximum score 0.42. The network $\mathcal{N}$ is fine tuned after grafting y_2^3 to achieve

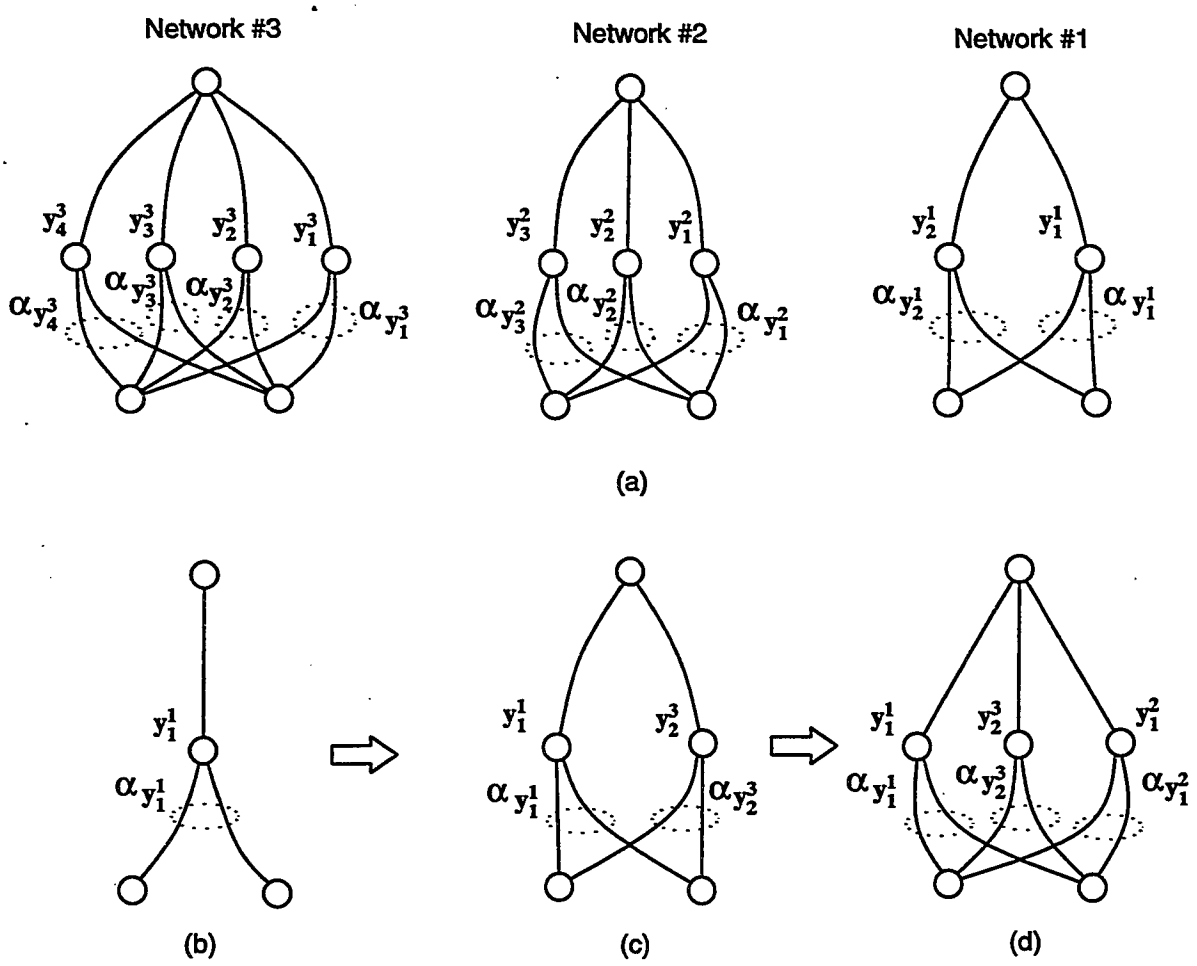


Figure 4.3: Illustration of the grafting example, (a) The three initial networks, (b) Network $\mathcal{N}$ after 1st iteration, (b) Network $\mathcal{N}$ after 2nd iteration, (d) Network $\mathcal{N}$ after 3rd iteration.

1st iteration

State:

$\Gamma = \{y_1^1, y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_2^3, y_3^3, y_4^3\}$, $\gamma = \{\}$

HN	y_1^1	y_2^1	y_1^2	y_2^2	y_3^2	y_1^3	y_2^3	y_3^3	y_4^3
Score	0.26	0.17	0.01	0.21	0.15	0.13	0.05	0.22	0.24

Action:

$\Gamma = \{y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_2^3, y_3^3, y_4^3\}$, $\gamma = \{y_1^1\}$

After fine tuning $\mathcal{N}(I, T, \gamma)$: Accuracy($\mathcal{N}$) = 0.5

2nd iteration

State:

$\Gamma = \{y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_2^3, y_3^3, y_4^3\}$, $\gamma = \{y_1^1\}$

HN	y_2^1	y_1^2	y_2^2	y_3^2	y_1^3	y_2^3	y_3^3	y_4^3
Score	0.23	0.25	0.26	0.30	0.33	0.42	0.17	0.27

Action:

$\Gamma = \{y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_3^3, y_4^3\}$, $\gamma = \{y_1^1, y_2^3\}$

After fine tuning $\mathcal{N}(I, T, \gamma)$: Accuracy($\mathcal{N}$) = 0.70

3rd iteration

State:

$\Gamma = \{y_2^1, y_1^2, y_2^2, y_3^2, y_1^3, y_3^3, y_4^3\}$, $\gamma = \{y_1^1, y_2^3\}$

HN	y_2^1	y_1^2	y_2^2	y_3^2	y_1^3	y_3^3	y_4^3
Score	0.50	0.71	0.22	0.41	0.37	0.34	0.10

Action:

$\Gamma = \{y_2^1, y_2^2, y_3^2, y_1^3, y_3^3, y_4^3\}$, $\gamma = \{y_1^1, y_2^3, y_1^2\}$

After fine tuning $\mathcal{N}(I, T, \gamma)$: Accuracy($\mathcal{N}$) = 1

Figure 4.4: Illustration of the grafting example.

accuracy of 0.70. The same process of selecting (of node y_1^2) and grafting is followed in the 3rd iteration. The network $\mathcal{N}$ achieved accuracy of 1 (perfect classification) after the 3rd iteration. This last network is reported as the final result in this example.

The above grafting algorithm is outlined in Figure 4.5. The input to **GRAFT** are a set $\mathcal{S}$ of training examples along with R trained networks each having I input nodes, T output nodes and some arbitrary number of hidden nodes. Each of the R networks is assumed to be independently trained until its error can not be made any smaller or MAX iterations have been reached. In Steps 1 and 2, the algorithm initializes Γ as the sets of all hidden nodes in the R networks and γ to be empty. In Step 3.1., the hidden node Best_y is selected from Γ whose inclusion in γ maximizes the classification accuracy of $\mathcal{N}(I,T,\gamma)$. This step is performed by training the network $\mathcal{N}(I,T,\gamma \cup \{y\})$ over the training examples ($\mathcal{S}$). During training, the weights between the output and hidden nodes (γ) are randomly initialized and then updated while freezing the weights between the hidden (γ) and input nodes to make the network $\mathcal{N}$ as consistent as possible with the training examples. Once, the hidden node Best_y is selected, it is removed from Γ and added to γ. In Step 3.3., the network $\mathcal{N}(I,T,\gamma)$ is fine tuned by applying BP learning algorithm for a fixed number of iterations. During fine tuning all weights are subject to change. The process of selection, adding and fine tuning is repeated several times until the network $\mathcal{N}$ reaches a satisfactory level of consistency or the number of hidden nodes, $|\gamma|$, reaches

Algorithm GRAFT (input: R networks, the training examples S):

1. Let $\Gamma = Y_1 \cup \cdots \cup Y_r \cup \cdots \cup Y_R$ where Y_r is the set of hidden nodes in the rth network.
2. $\gamma = \{\}$.
3. Repeat until (Accuracy($\mathcal{N}(I, T, \gamma)$) $> (1 - \epsilon)$) or ($|\gamma| >$ *Limit*)

3.1. $\text{Best}_y = y$ such that Accuracy($\mathcal{N}(I, T, \gamma \cup \{y\})$) is maximal over all $y \in \Gamma$.

3.2. $\Gamma = \Gamma - \{\text{Best}_y\}, \gamma = \gamma \cup \{\text{Best}_y\}$.

3.3. Fine-tune $\mathcal{N}(I, T, \gamma)$ (train using BP for a fixed number of iterations).

4. Report $\mathcal{N}(I, T, \gamma)$.

Figure 4.5: Outline of the grafting algorithm

Limit where *Limit* is an experimental condition for the upper number of hidden nodes to be grafted.

Chapter 5

Simulations and Comparisons

5.1 Introduction

Two types of problems, function approximation and classification, were simulated using the grafting algorithm discussed in Chapter 4. For the function approximation, we have chosen to simulate several instances of the parity and counter problems. The parity and counter problems have been used in the literature to test the performance of learning algorithms [1, 26, 23, 14]. For the classification problem, we have chosen the glass classification [20] as a natural domain.

To compare our results to other approaches, we have chosen two methods, one representing the pruning and the other representing the dynamic construction approach. In addition, the same problems were solved using networks with predetermined sizes. From the pruning approach, we used the node pruning method; and

from the dynamic construction, we used the dynamic node creation method. The classical BP training algorithm was applied in training for the four approaches.

The BP algorithm used in the comparisons is described earlier in Section 2.2. In Section 5.2.1 and 5.2.2, we will describe the details of the node pruning algorithm and the dynamic node creation algorithm used in the comparisons, respectively. In Sections 5.3.1 and 5.3.2, we will present the simulation and comparisons results of the parity, and counter problems, respectively. In Section 5.3.3, we will present the simulation and comparisons results of the glass classification problem.

5.2 Selected Algorithms

5.2.1 Node Pruning

As a representative of the pruning approach, we have selected the node pruning method to use in our experiments. This method is a variation of the first pruning approach presented by Sietsma and Dow [27], which is based on pruning hidden nodes that have least contribution to the network. The contribution of a node is computed by measuring the actual error caused by pruning that node.

The node pruning procedure implemented here is outlined in Figure 5.1. A three layer network, I-J-T, with a large number of hidden nodes, J, is trained using BP over the training examples S until the error can not be made any smaller. In Step 2.1., the accuracy of the trained network is reported. Then, the node that

1. Train network I-J-T where $J = 10$ using BP over S
   until the error is within ϵ.
2. While $(J > 0)$ repeat
2.1. Report the accuracy of the network I-J-T.
2.2. Select hidden node j where Error(j) is minimum among all
   hidden nodes.
2.3. Prune the network by eliminating j and modify the architecture
   from I-J-T to I-$(J-1)$-T and set $J = J - 1$.
2.4. Retrain the network after pruning using BP to recover.
3. Report the accuracy of the last network.

Figure 5.1: Outline of the node pruning algorithm.

causes least distortion if pruned is selected. The distortion is calculated by

$$\text{Error}(j) = E_{\text{without j}} - E_{\text{with j}} \cdot$$

After selecting hidden node j, it is pruned and the architecture of the network is changed from I-J-T to I-$(J-1)$-T. After that, the network is given a chance to recover by further training. This loop of selection and pruning is continued until the number ofhidden nodes reachs 1.

5.2.2 Dynamic Node Creation

From the dynamic construction approach, we have chosen the dynamic node creation method [1]. It is based on adding hidden nodes with random weights to a network when the error does not fall below an error of ϵ or after MAX iterations have been reached.

The dynamic node creation algorithm that we have implemented is outlined in Figure 5.2. Initially, a network I-2-T is trained until the error can not be made any smaller over the training examples S or until MAX iterations have been performed. A new node is then added to the hidden layer. The weights between the added node and the input and output nodes are initialized with random values. The network is then trained after adding the new hidden node. The addition of the hidden node changes the network architecture from I-J-T to I-$(J+1)$-T. The loop of adding and training is repeated several times until the number of hidden nodes reachs 10.

5.3 Simulations

5.3.1 Parity Experiments

The grafting algorithm was used to solve three parity problems, namely the xor, 5-bit and 6-bit parity. When the network is given the input vector it tells what the parity bit should be. For every experiment, there are 2^n examples where n is length

1. Train network I-2-T for using BP over S until the error can not be made any smaller or for MAX number of iterations and set $J = 2$.
2. While $(J < 10)$ repeat
   2.1. Change the network from I-J-T to I-$(J+1)$-T by adding one node with random incoming and outgoing weights and set $J = J + 1$.
   2.2. Retrain the network using BP after the addition.
3. Report last constructed network.

Figure 5.2: Outline of the dynamic node creation algorithm.

of the input vector. In the parity experiments, all the input values are binary (0 or 1). The output is considered 1 when it is θ away from the desired output where is θ is a real number between 0 and 1. For example, if $\theta = 0.5$, then an output of 0.51 or more is considered 1 and output of 0.49 or less is considered 0. In this experiment, θ is set to 0.3. We used Eq(2.19) to compute the classification accurecy over all examples. For every experiment, we start with a small network of one hidden layer with 1 hidden node, I input nodes, and 1 output node, I-1-1. Then, the grafting algorithm is applied to graft from the hidden nodes of previously R trained networks until the accuracy is 100%. The weights are initialized with random values in the interval $[-3, 3]$.

Table 5.1 gives the parameters used to graft the target networks for the parity

Table 5.1: Details of parameters used in the parity problems.

Problem	I	T	# Examples	R	$\|\Gamma\|$	$iter_1$	$iter_2$
xor	2	1	4	5	15	30	600
5-bit	5	1	32	9	54	30	600
6-bit	6	1	64	10	45	30	600

experiment. Each network consists of I input nodes and T output nodes. The number of the networks used to graft is R and the number of all hidden nodes for the R networks is $|\Gamma|$. The target network is trained $iter_1$ iterations while selecting which node to graft and $iter_2$ iterations during fine tuning. The classification accuracy vs hidden nodes of the target networks for the three parity problems are depicted in Figures 5.3, 5.5, and 5.7. The average square error after $iter_2$ iterations vs number hidden nodes for the target networks is shown in Figures 5.4, 5.6, and 5.8.

In Figures 5.3 and 5.4, the four techniques exhibit very similar behavior for the xor problem for all numbers of hidden nodes. It is so because of the simple nature of the problem. For the 5-bit parity, the grafting algorithm performs better accuracy than the other approaches when the number of hidden nodes were 2 and 3, and close accuracy to the predetermined network size approach and node pruning at 4 and 5. For 6 hidden nodes and more, all the four approaches have shown little difference in accuracy as it is close to 100%.

In the 6-bit parity experiment, the accuracy of the grafting algorithm outperforms the other three approaches when the number of hidden nodes is between 1 and 5. When the number of hidden nodes is 6 or 7, the grafting algorithm achieves

better results than pruning and dynamic node creation but slightly worse than the predetermined network size approach. The reason of the previous note is that the results of the the predetermined network size approach are the best out of many runs and randomness plays a role in the training process. All the four approaches achieve perfect classification when the number of hidden nodes is 8 or more.

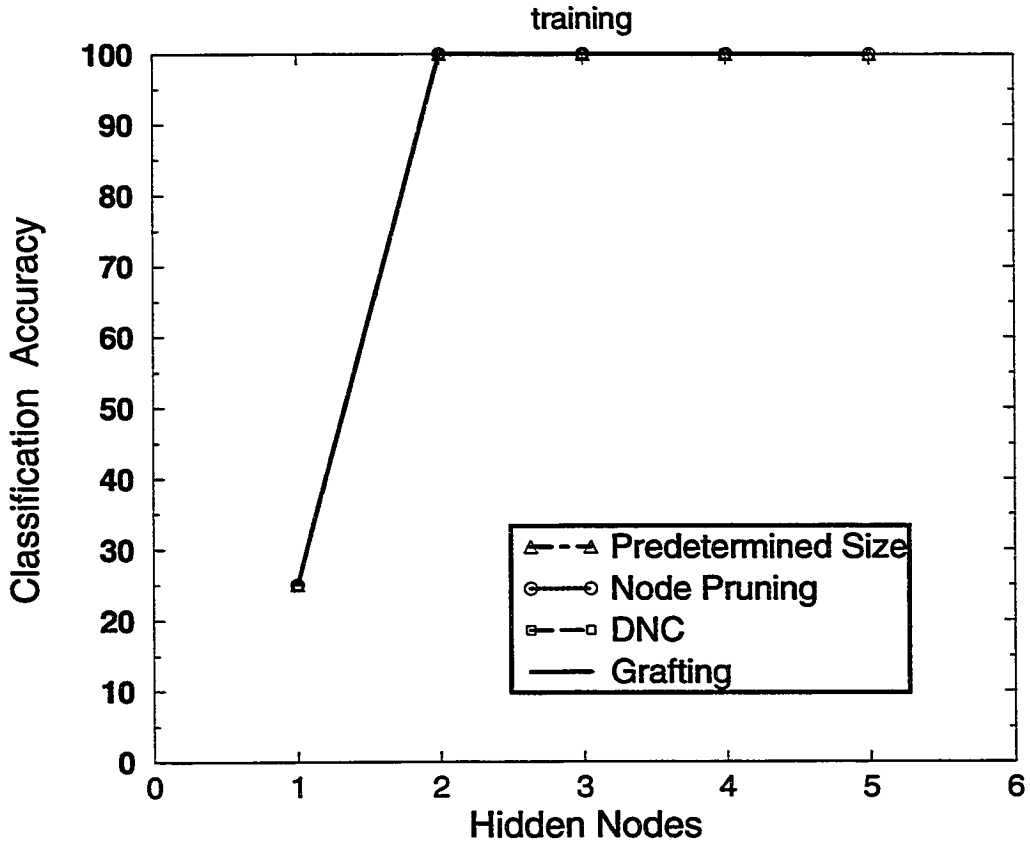


Figure 5.3: Classification Accuracy of training vs number of hidden nodes for the xor.

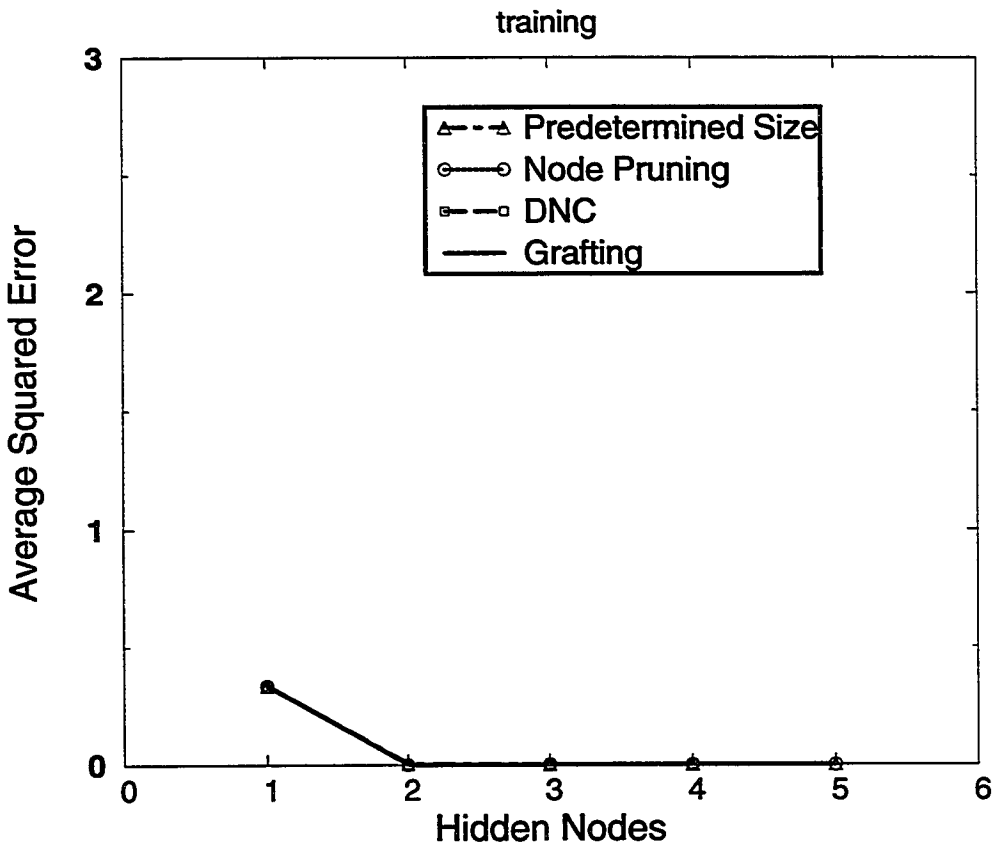


Figure 5.4: Average squared error of training vs number of hidden nodes for the xor.

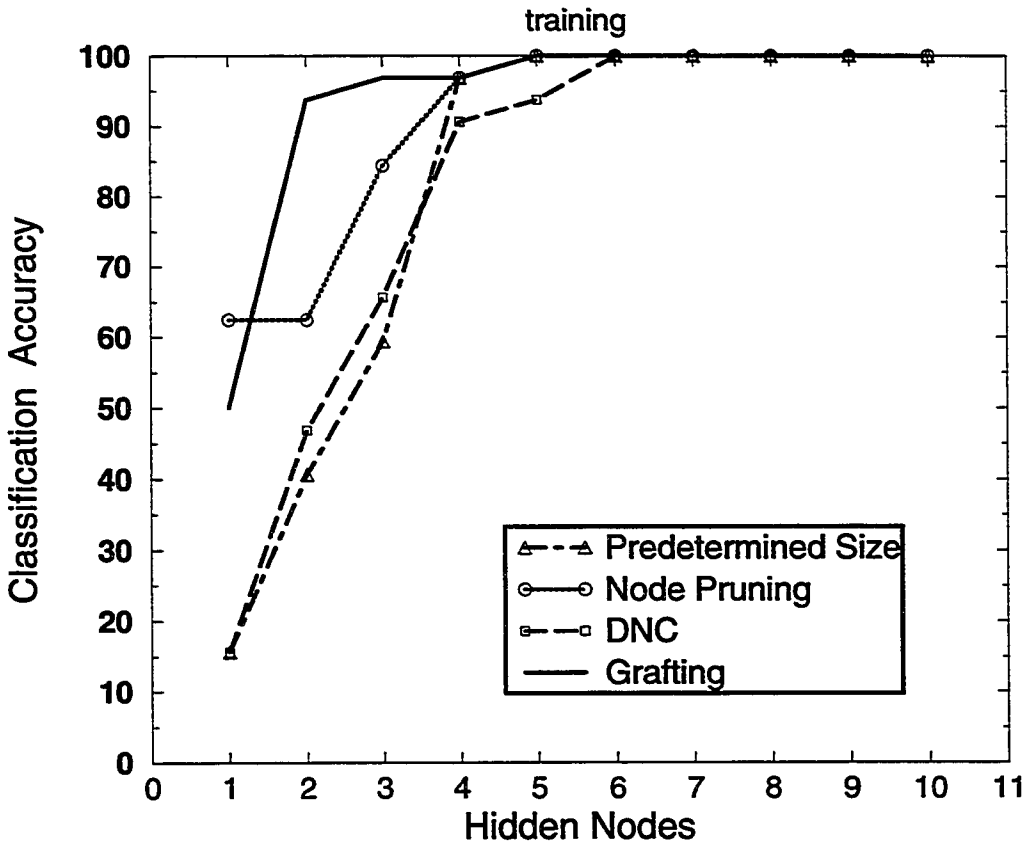


Figure 5.5: Classification Accuracy of training vs number of hidden nodes for the 5-bit parity.

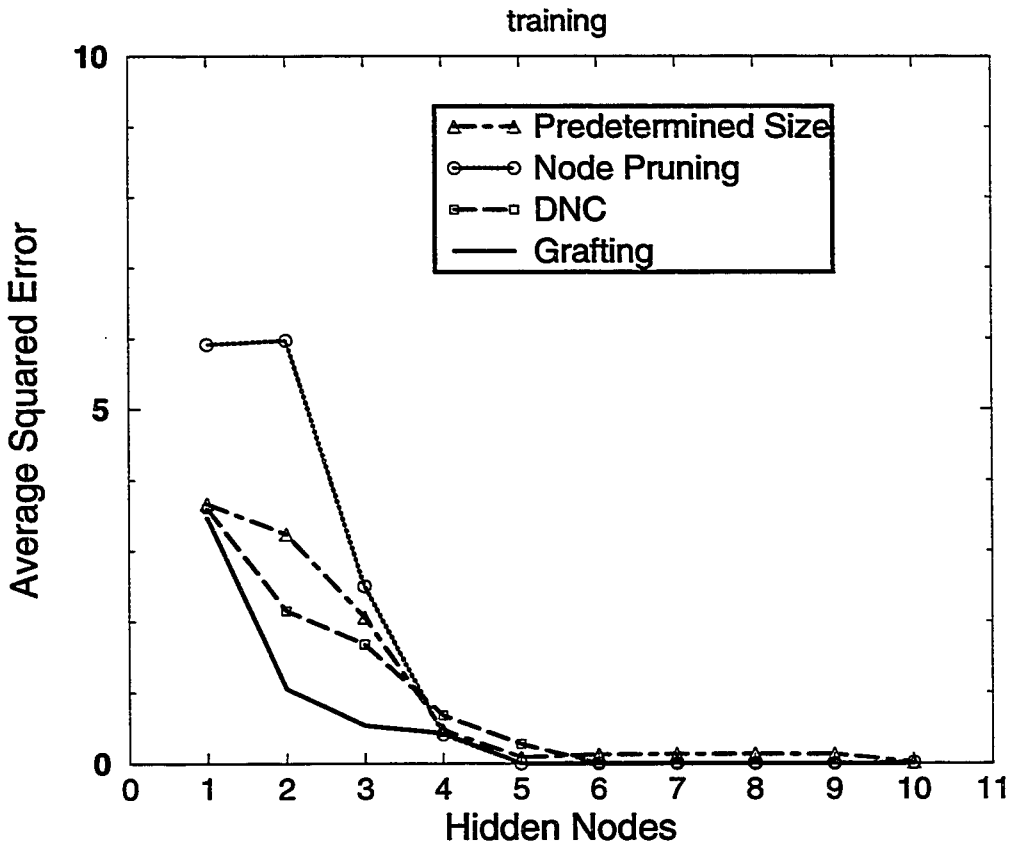


Figure 5.6: Average squared error of training vs number of hidden nodes for the 5-bit parity.

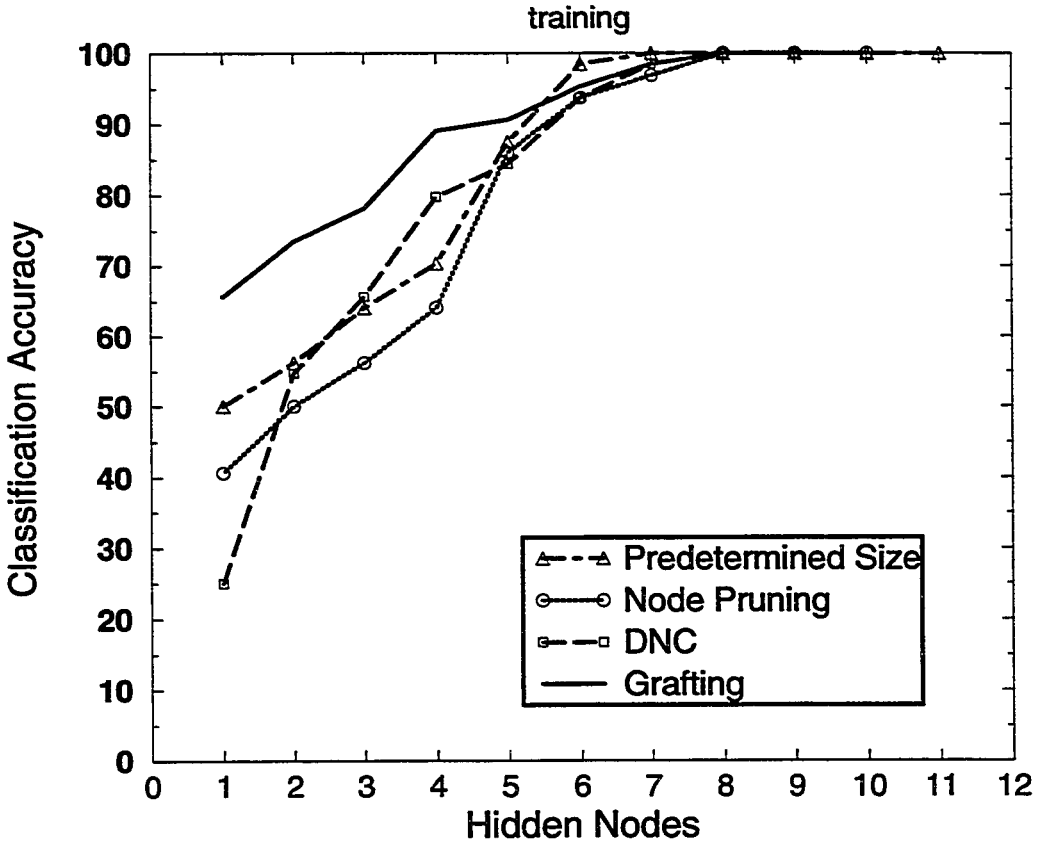


Figure 5.7: Classification Accuracy of training vs number of hidden nodes for the 6-bit parity.

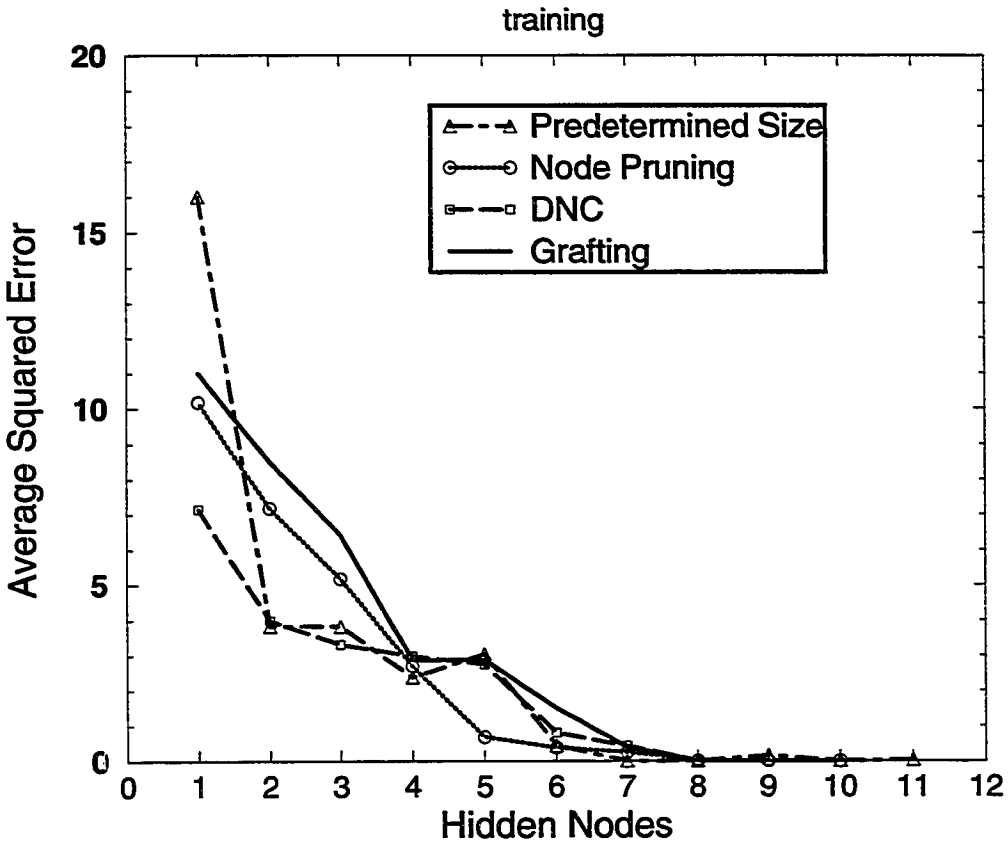


Figure 5.8: Average squared error of training vs number of hidden nodes for the 6-bit parity.

5.3.2 Counter Experiment

Three counter problems were solved using our grafting algorithm, namely 4-bit, 5-bit, and 6-bit. The network is supposed to increment the given input vector by one. For example, suppose that the input vector is $< 1011 >$, then the network output vector should be $< 01100 >$. For every experiment, there are 2^n examples where n is length of the input vector. In the counter experiments, all the input values are binary (0 or 1). The output is considered 1 when it is θ away from the desired output. For example, if $\theta = 0.4$, then an output of 0.61 or more is considered 1 and output of 0.39 or less is considered 0. In this experiment, θ is set to 0.3. We used Eq(2.19) to compute the classification accurecy over all examples. For every experiment, we start with small network I-1-$(I+1)$ where I is the number of input nodes, $(I+1)$ is the number of output nodes, and one hidden node. Then, when the grafting algorithm is applied to graft from the hidden nodes of previously R trained networks until the accuracy is 100% or the number of grafted hidden nodes reaches *Limit*. The weights are initialized with random values in the interval $[-3, 3]$.

Table 5.2 gives the details of the parameters used to graft the target networks for the counter experiments. Each network consists of I input nodes and T output nodes. The number of the networks used to graft is R and the number of all hidden nodes for the R networks is $|\Gamma|$. The target network is trained $iter_1$ iterations while selecting which node to graft and $iter_2$ iterations during fine tuning. The

Table 5.2: Details of parameters used in the counter problems.

Problem	I	T	# Examples	R	$\|\Gamma\|$	iter_1	iter_2
4-bit	4	5	16	7	49	30	600
5-bit	5	6	32	9	91	30	600
6-bit	6	7	64	11	88	30	600

classification accuracy vs hidden nodes of the target networks for the three counter problems are depicted in Figures 5.9, 5.11, and 5.13. The average square error after iter_2 iterations vs number hidden nodes for the target networks is shown in Figures 5.10, 5.12, and 5.14.

In all 4, 5, and 6-bit counters, the grafting algorithm achieves classification performance better than the other three approaches as shown in Figures 5.9, 5.11, and 5.13. This suggests that as the number of hidden nodes increases, the four approaches have the same performance which suggests that the performance reaches its peak and can not be improved.

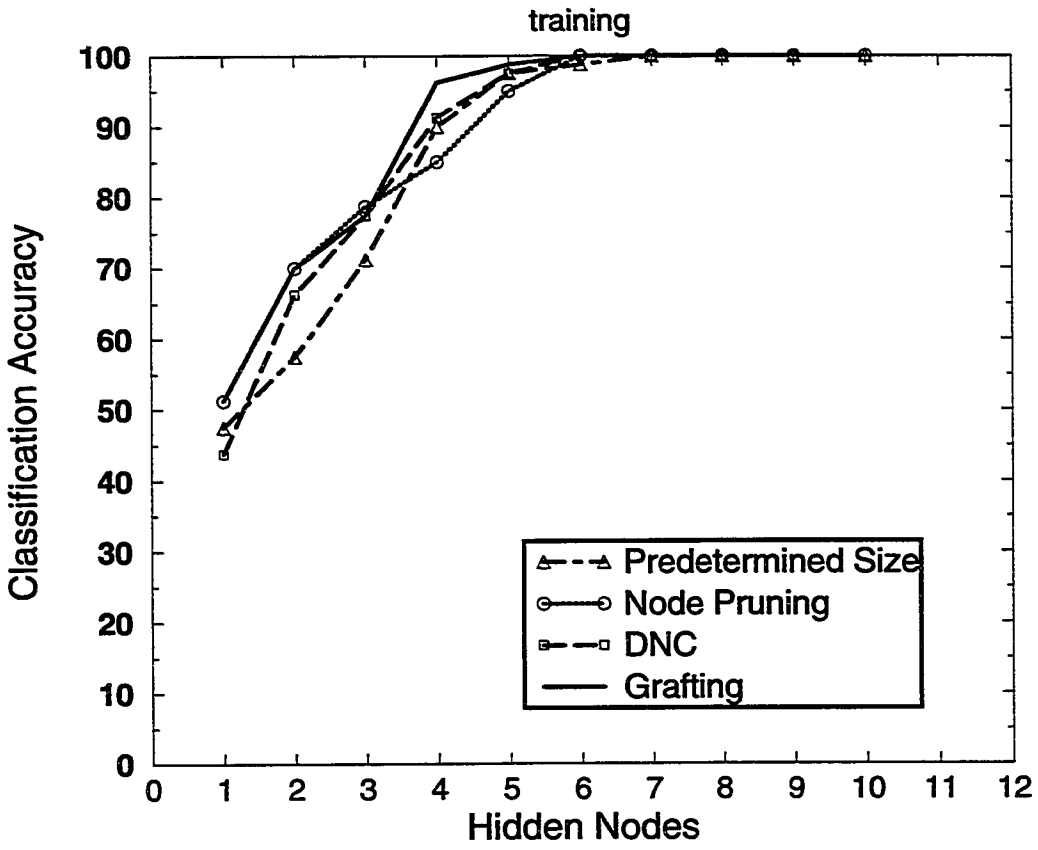


Figure 5.9: Classification Accuracy of training vs number of hidden nodes for the 4-bit counter.

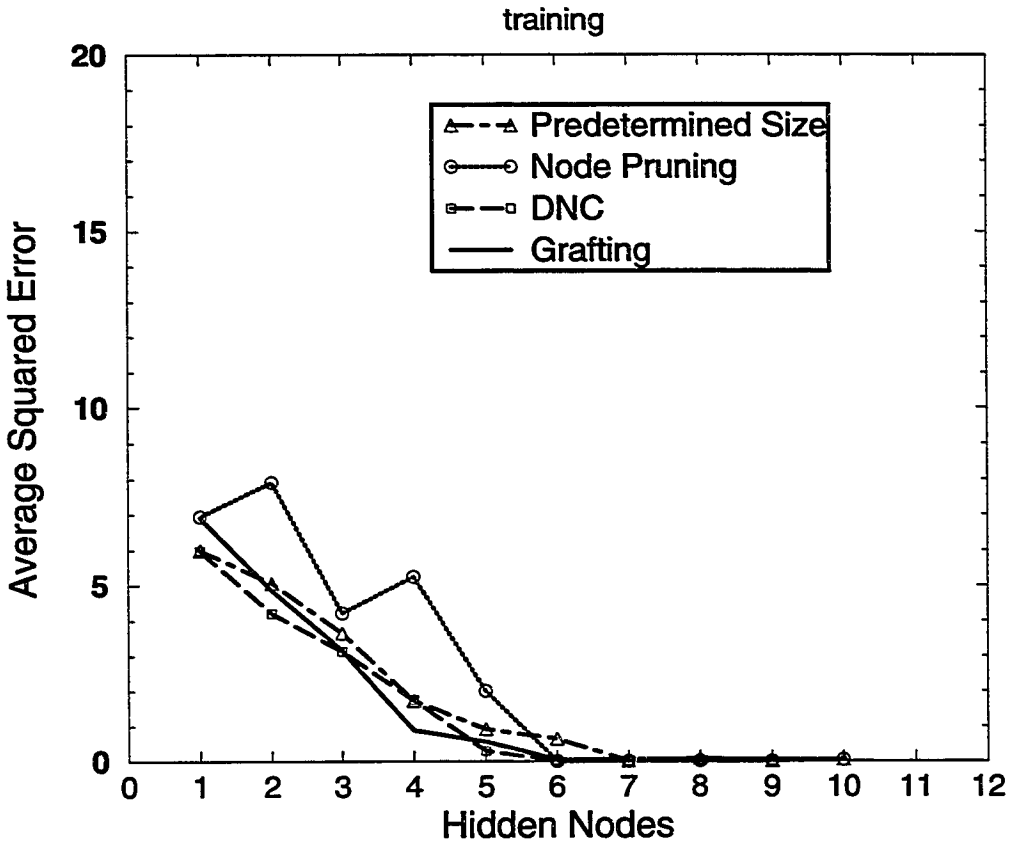


Figure 5.10: Average squared error of training vs number of hidden nodes for the 4-bit counter.

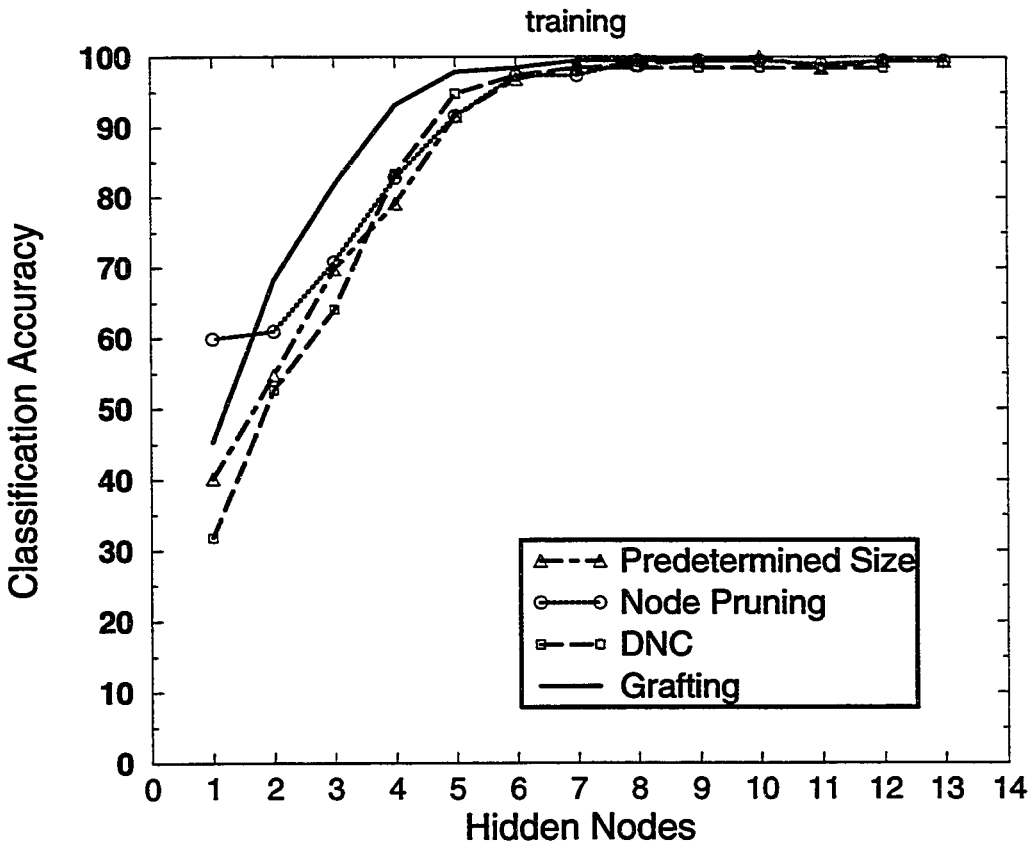


Figure 5.11: Classification Accuracy of training vs number of hidden nodes for the 5-bit counter.

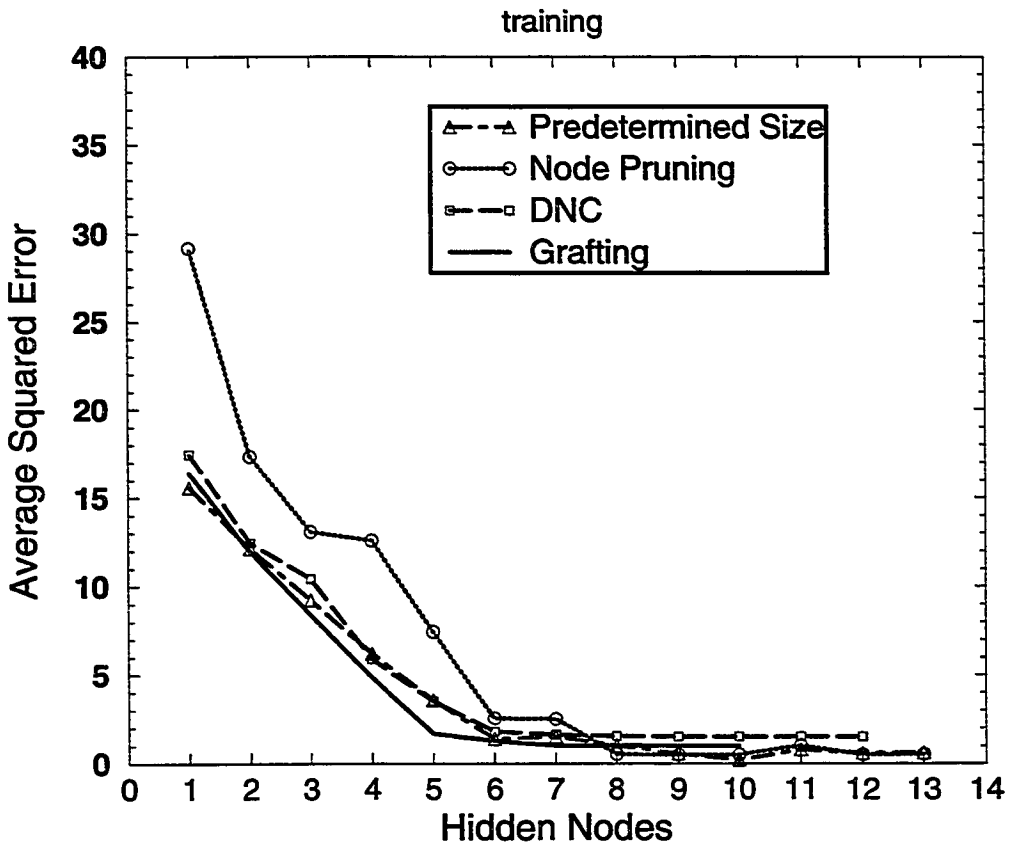


Figure 5.12: Average squared error of training vs number of hidden nodes for the 5-bit counter.

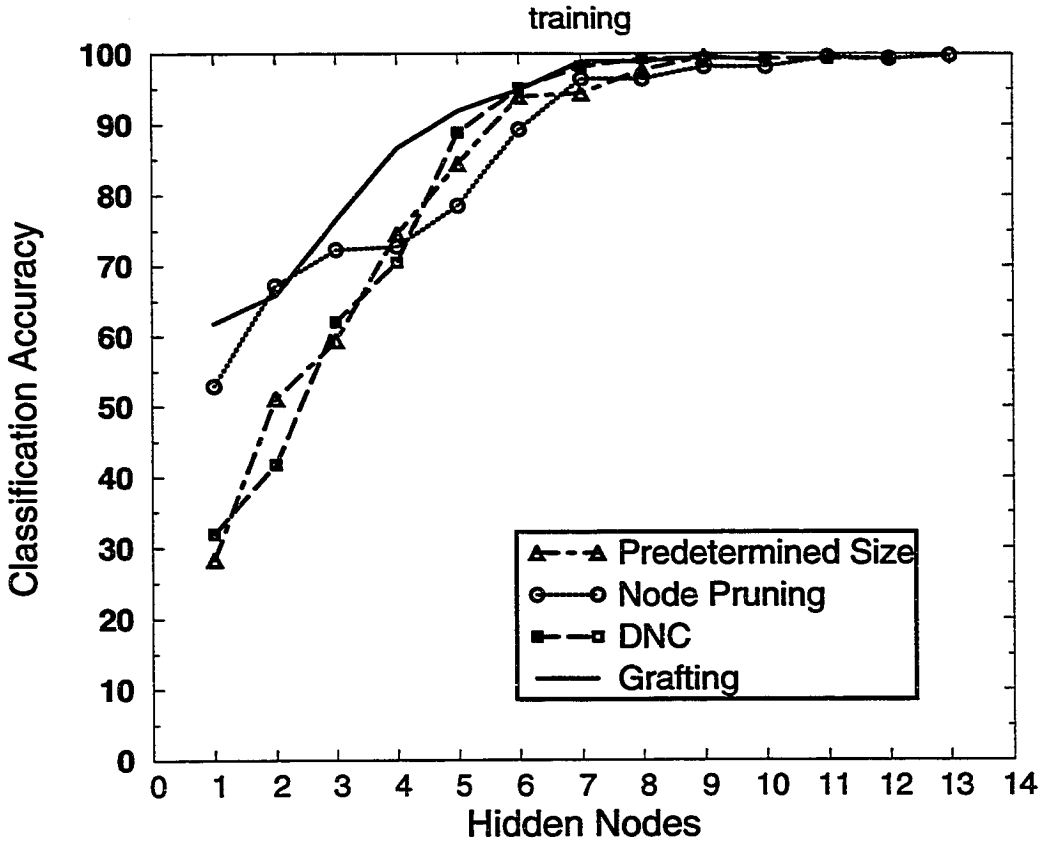


Figure 5.13: Classification Accuracy of training vs number of hidden nodes for the 6-bit counter.

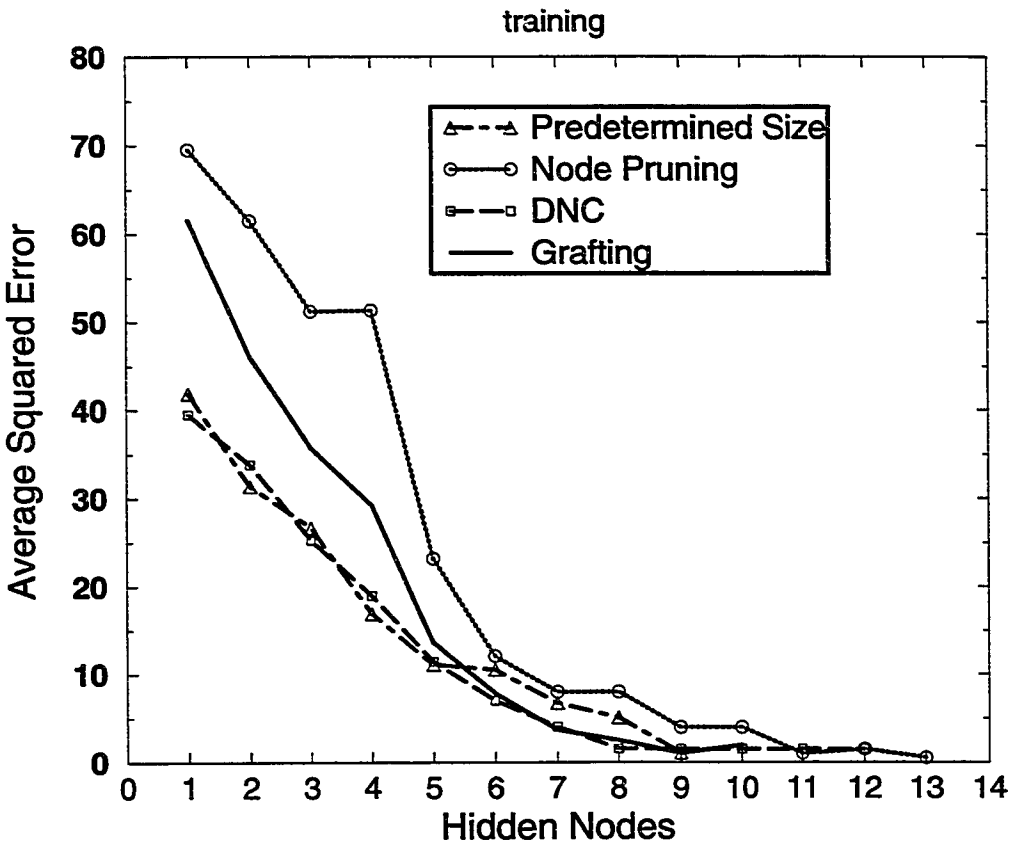


Figure 5.14: Average squared error of training vs number of hidden nodes for the 6-bit counter.

Table 5.3: Attribute Information

Attribute	Description
1	RI: refractive index
2	Na: Sodium
3	Mg: Magnesium
4	Al: Aluminum
5	Si: Silicon
6	K: Potassium
7	Ca: Calcium
8	Ba: Barium
9	Fe: Iron

5.3.3 Glass Classification Experiment

In this section, we describe our experiment on a natural domain called Glass Classification which is taken from [20]. The chemical properties of the manufactured glass depend on its use. For example, window glass has different chemical elements concentrations that tableware or head lamp glass. Therefore, if we have a sample piece of glass and analyze its chemical properties, we can then suggest the suitable application for that piece of glass or know from where it came. In our experiment, we know 9 features that can be used to classify the type of the glass. These features are listed in Table 5.3. The number of instances available in this domain is 214. We have used 100 instances for training and 114 for testing. The 214 instances are classified into 7 classes based on their use. The class distribution is summarized in Table 5.4. A statistical summary of each feature is shown in Table 5.5. The details of the parameters used to graft the target network are given in Table 5.6. Each network consists of I input nodes and T output nodes. The number of the networks

Table 5.4: Glass Distribution.

# Instances	Class Description	Class Code
70	float building windows	1 1 1 0
17	float vehicle windows	1 0 0 1
76	non-float building windows	1 1 1 1
0	non-float vehicle windows	0 0 0 0
13	containers	1 0 1 1
9	tableware	0 1 1 1
29	head lamps	1 1 0 1

Table 5.5: Statistical Summary of the glass domain.

Feature	Attribute	Min	Max	Mean	SD	Correlation
1	RI	1.5112	1.5339	1.5184	0.0030	-0.1642
2	Na	10.73	17.38	13.4079	0.8166	0.5030
3	Mg	0	4.49	2.6845	1.4424	-0.7447
4	Al	0.29	3.5	1.4449	0.4993	0.5988
5	Si	69.81	75.41	72.6509	0.7745	0.1515
6	K	0	6.21	0.4971	0.6522	-0.0100
7	Ca	5.43	16.19	8.9570	1.4232	0.0007
8	Ba	0	3.15	0.1750	0.4972	0.5751
9	Fe	0	0.51	0.0570	0.0974	-0.1879

used to graft is R and the number of all hidden nodes for the R networks is $|\Gamma|$. The target network is trained $iter_1$ iterations while selecting which node to graft and $iter_2$ iterations during fine tuning. The classification accuracy vs hidden nodes of the target networks for the training and testing examples are depicted in Figures 5.15, and 5.17, respectively. The average square error after iter_2 iterations vs number of hidden nodes for the target networks for the training and testing examples are shown in Figures 5.16, and 5.18, respectively.

The performance of the grafting algorithm in the glass classification problem

Table 5.6: Details of parameters used in the glass classification problem.

Problem	I	T	Training Exmp's	Testing Exmp's	R	$\|\Gamma\|$	$iter_1$	$iter_2$
glass Class.	8	4	100	114	13	91	10	600

Table 5.7: Classification Accuracy at saturation for the glass classification.

Appraoch	HN	Class. Acc.	
		Training	Testing
Pred. Size	9	88.28	71.49
Node Pruning	12	87.25	73.90
DNC	9	89.50	72.59
Grafting	3	88.25	73.46

for the classification accuracy is shown in Figures 5.15 and 5.17. The classification accuracy for the training examples outperforms the other approaches as it reachs saturation when the number of hidden nodes is 3. The other approaches need up to 9 hidden nodes to reach saturation. As the number of hidden nodes increases, performance tends to be almost constant. On the other hand, the performance on the testing examples is significantly better when the number of hiddens is 3 than the other approaches for the first 8 hidden nodes. As the number of hidden nodes increases, the performance has little difference among the networks built by the four approaches. The above suggests that the grafting algorithm enhances the network ability to generalize from the early grafting iterations. Table 5.7 summarizes the classification accuracy for the training and testing examples for the four approaches at saturation. The earliest best achievements of grafting is 88.25 for training and 73.46 for testing when the number of hidden nodes is 3. The other approaches reach

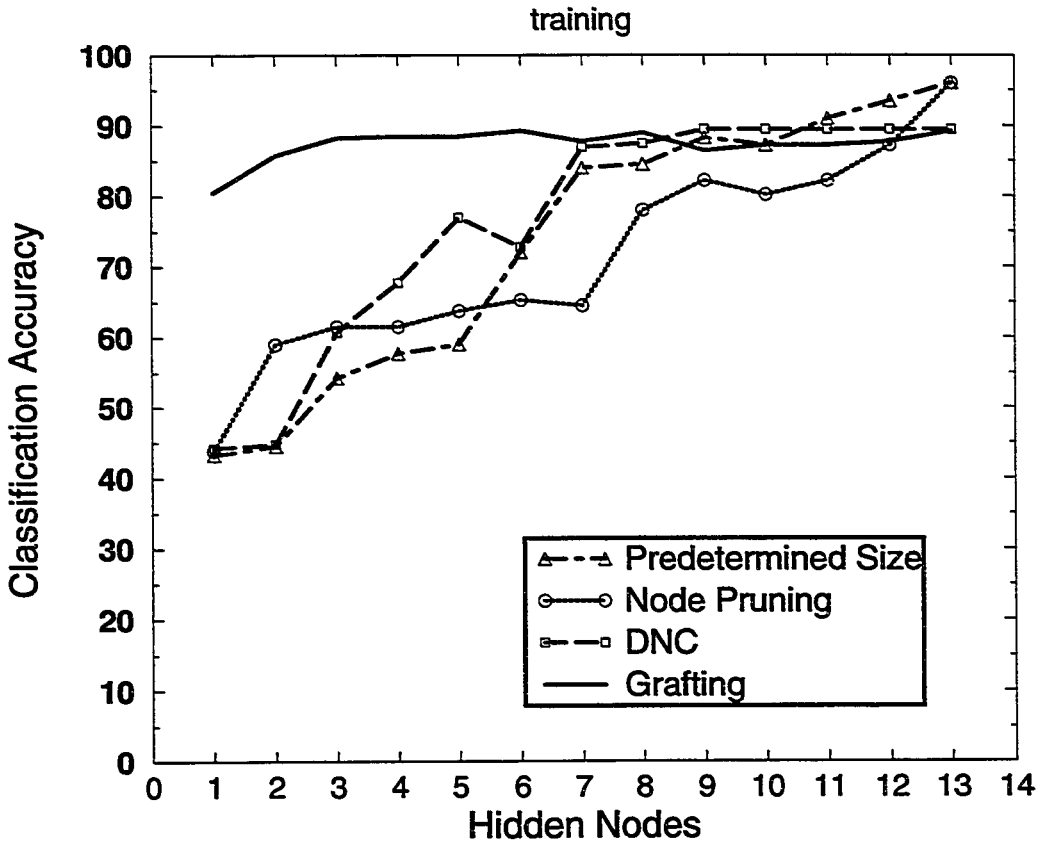


Figure 5.15: Classification Accuracy of training vs number of hidden nodes for the glass classification.

this level of classification accuracy at 9, 12, and 9 for predetermined network size, node pruning and dynamic node creation, respectively.

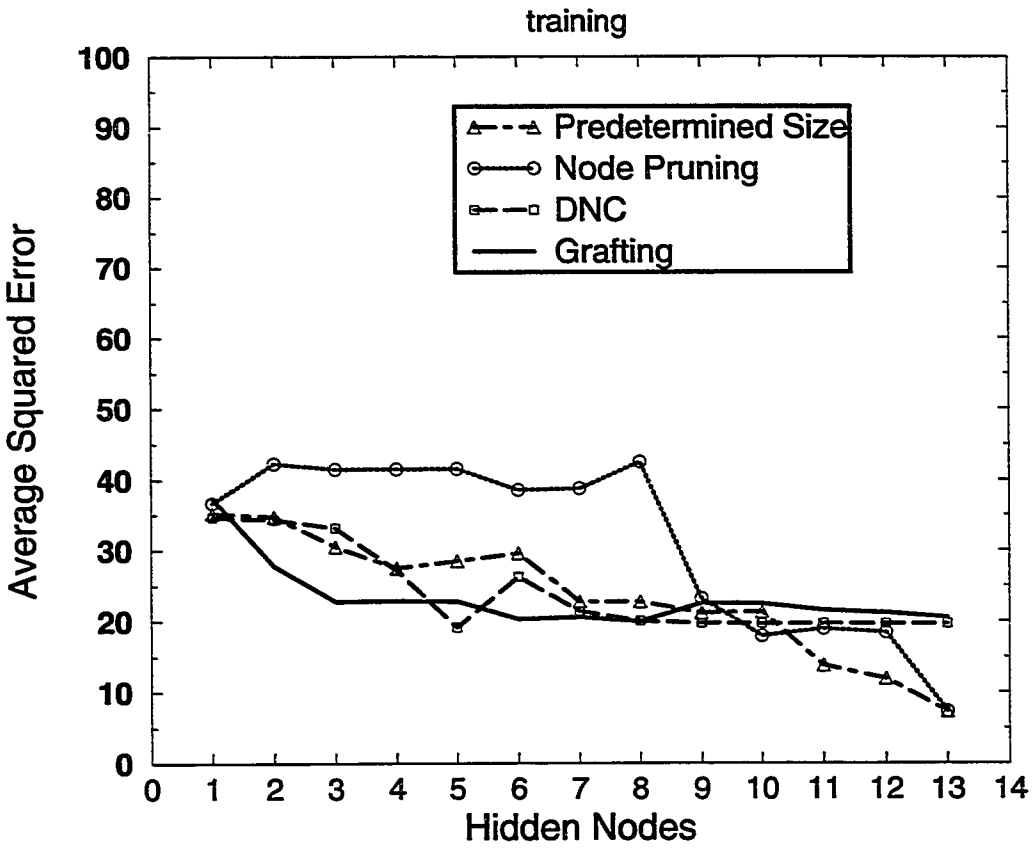


Figure 5.16: Average squared error of training vs number of hidden nodes for the glass classification.

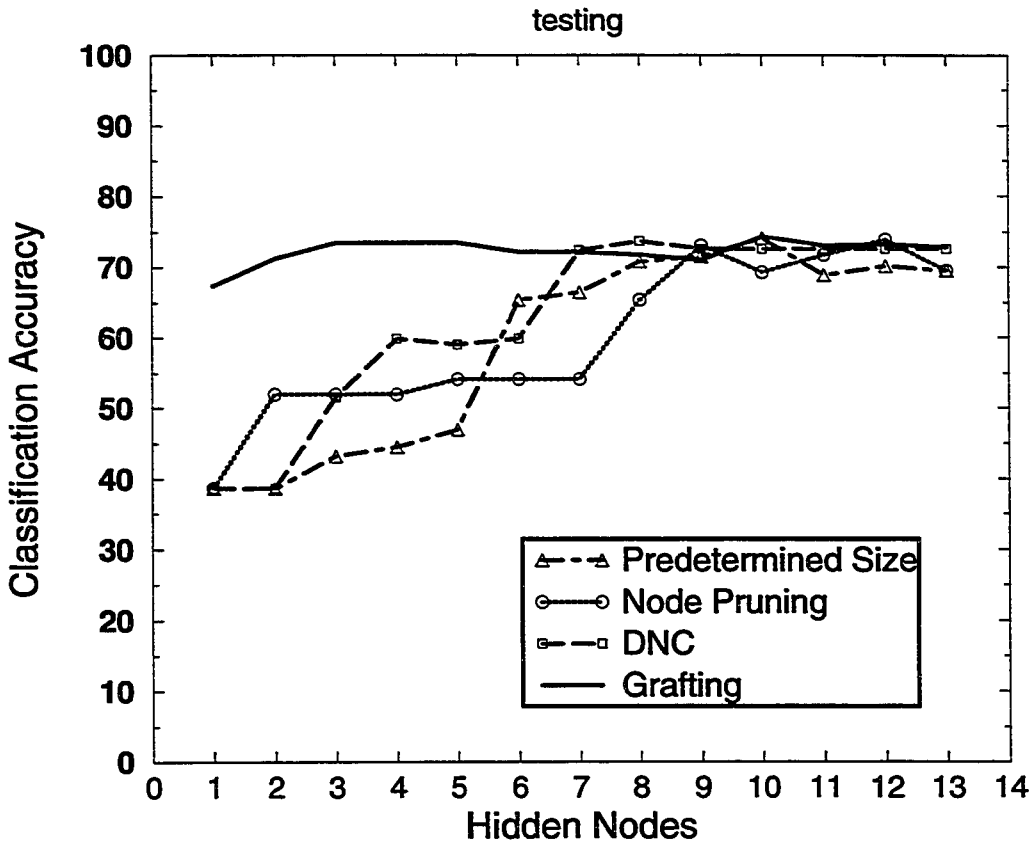


Figure 5.17: Classification Accuracy of testing vs number of hidden nodes for the glass classification.

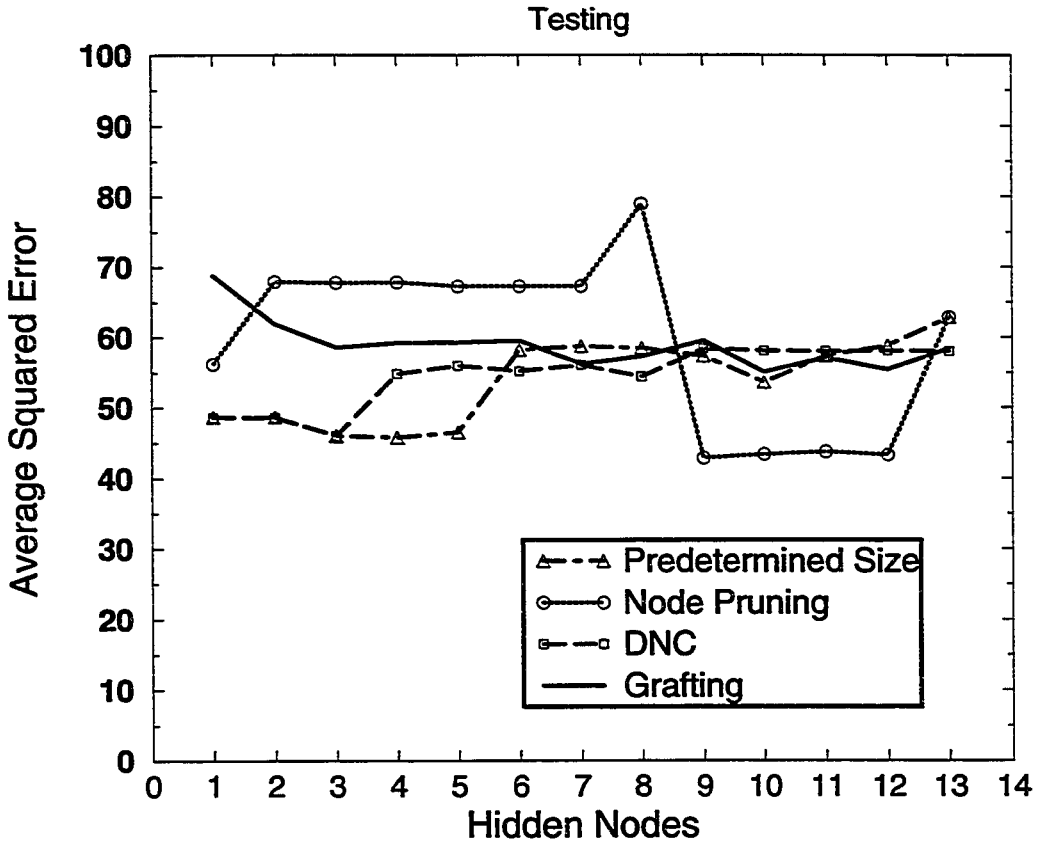


Figure 5.18: Average squared error of testing vs number of hidden nodes for the glass classification.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this research, we have reviewed several issues concerned with learning in feedforward neural networks. In particular, the thesis focuses on the connection between the size of a network and its ability to "memorize" and "generalize". There are no known methods to determine the exact number of hidden nodes that are necessary and sufficient to solve a given problem. Previous techniques that address this problem fall into two approaches, namely pruning and dynamic construction. These approaches try to find the minimal number of hidden nodes by either starting with a large network and shrinking it down by pruning its components as in the pruning approach, or by starting with a small network and then enlarging it by adding components as in the dynamic construction approach. The main limitation of these

approaches is that if the training does not yield a network which is consistent with the training examples, then the process of training the network has to be repeated several times until such a network is built.

In this thesis, we have introduced a new approach to construct neural networks in which the history plays an important role. We have called our approach grafting because we graft hidden nodes from several trained networks by selecting those parts that appear to have captured relevant knowledge. The accomplishments of the thesis can be summarized as follows:

1. A new grafting algorithm was developed which follows a greedy strategy to graft neural networks.

2. The grafting algorithm was used to solve artificial as well as natural problems. From the artificial domain we used our grafting approach to solve three parity problems and three counter problems. The parity problems are the xor, 5-bit and 6-bit parities. The counter problem are the 4-bit, 5-bit and 6-bit counters. From the natural domain, we have selected the glass classification problem.

3. We have compared the results achieved of our grafting algorithm to:

   (a) Predetermined Network Size

   (b) Dynamic Node Creation

   (c) Node Pruning

In all the four approaches, back-propagation learning was used. Our grafting algorithm gave favorable results compared to the other approaches.

6.2 Future Work

The following points give interesting directions for future research:

1. Generalizing the grafting algorithm for a feedforward network of more than 3 layers.

2. Developing a weight grafting algorithm in which weights can be used to construct networks.

3. Investigating the employment of different learning algorithms other than back-propagation in grafting.

4. Changing the selection criteria from the greedy approach into a new one which would make use of the knowledge captured by hidden nodes.

5. Comparing our grafting algorithm to other techniques such as genetic based approaches [33, 24], and other pruning approaches [13, 28, 30].

References

[1] T. Ash. Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4):365–375, 1989.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *Journal of Associate Computing Machinary*, 36(4):929–965, 1989.

[3] P. Burrascano. A Pruning Technique for Maximizing Generalization. In *IEEE International Joint Conference on Neural Networks*, pages 347–350, 1993.

[4] Y.L. Cun, J.S. Denker, and S.A. Solla. Optimal Brain Damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, 1990.

[5] Y.L. Cun, I. Kanten, and S.A. Solla. Eigenvalues of Coveriance Matrices: Application to Neural-Network Learning. *Physical Review Letters*, 66(18):2396–2399, May 1991.

[6] K.I. Diamantaras and S.Y. Kung. Cross-Correlation Neural Network Models. *IEEE Transactions on Signal Processing*, 42(11):3218–3223, Nov. 1994.

[7] S.E. Fahlman and C. Lebiere. The Cascade-Correlation Learning Architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, 1990.

[8] M. Fukumi, S. Omatu, F. Takeda, and T. Kosaka. Rotation-Invariant Neural Pattern Recognition System with Application to Coin Recognition. *IEEE Transactions on Neural Networks*, 3(2):272–279, March 1992.

[9] S. J. Hanson and L.Y. Pratt. Comparing Biases for Minimal Network Construction with Back-Propagation. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 177–185, 1988.

[10] D.R. Hush and B. G. Horne. Progress in Supervised Neural Networks. *IEEE Signal Processing Magazine*, pages 8–39, Jan 1993.

[11] J.S. Judd. *Neural Network Design and the Complexity of Learning.* A Bradford Book, 1990.

[12] R. Kamimura. Principal Hidden Unit Analysis: Generation of Simple Networks by Minimum Entropy Method. In *IEEE International Joint Conference on Neural Networks*, pages 317–320, 1993.

[13] E.D. Karnin. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, June 1990.

[14] D. A. Karras and S. J. Perantonis. An Efficient Constrained Training Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, 6(6):1420–1434, Nov. 1995.

[15] S. Y. Kung. *Digital Neural Networks.* Prentic Hall, 1993.

[16] R. P. Lippmann. An Introduction to Computing with Neural Nets. *IEEE Acoustics, Speech and Signal Processing Magazine*, 4(2):4–22, Apr 1987.

[17] A. Luk, S.H. Leung, W.H. Lau, and S.L. Lee. Pruning via Number of Links and Neuron Activities. In *Prceedings of the IEEE International Symposium on Circuits and Systems*, volume 4, pages 2415–2418, 1993.

[18] P. Mehrotra, J.E. Quaicoe, and R. Venkastesan. Development of An Aritificial Neural Network Based Induction Motor Speed Estimator. In *Proceedings of Power Electronics Specialists*, Baveno, Italy, June 1996.

[19] M.C. Mozer and P. Smolensky. Skeletonization : A Technique for Trimming the Fat from a Network via Relevance Assessment. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 107–115, 1989.

[20] P.M. Murphy and D.W. Aha. UCI Repository of machine learning databases. [Machine-readable data repository]. Irvine, CA: University of California, Department of Information and Computer Science, 1993.

[21] D. Parker. Learning Logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, 1985.

[22] R. Reed. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, Sep. 1993.

[23] D.E. Rumelhart, G.E. Hinton, and R.J. Wiliams. Learning Internal Representations by Error Propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processings*, volume 1. MIT Press, 1986.

[24] W. Schiffmann, M. Joost, and R. Werner. Synthesis and Performance Analysis of Multilayer Neural Network Architectures. Technical report, University of Koblenz, Institute of Physics, 1992.

[25] W. Schiffmann, M. Joost, and R. Werner. Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons. Technical report, University of Koblenz, Institute of Physics, 1994.

[26] R. Setiono and L. C. K. Hui. Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm. *IEEE Transactions on Neural Networks*, 6(1):273–277, Jan. 1995.

[27] J. Sietsma and R. J. F. Dow. Neural Net Pruning-Why and How. In *IEEE International Joint Conference on Neural Networks*, volume I, pages 325–333, 1988.

[28] Z. Wang, C.D. Massimo, M.T. Tham, and J Morris. A Procedure for Determining the Topology of Multilayer Feedforward Neural Networks. *Neural Networks*, 7(2):291–300, 1994.

[29] E. Watanabe and H. Shimizu. Algorithm for Pruning Hidden Units in Multi Layered Neural Network for Binary Pattern Classification Problem. In *IEEE International Joint Conference on Neural Networks*, pages 327–330, 1993.

[30] A.S. Weigend, D. E. Rumelhart, and B.A. Huberman. Generalization by Weight-Elimination Applied to Currency Exchange Rate Prediction. In *IEEE International Joint Conference on Neural Networks*, volume 1, pages 837–841. Seattle, July 1991.

[31] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Generalization by Weight-Elimination with Application to Forecasting. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 3, pages 875–882, 1991.

[32] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavvior Science.* PhD thesis, Harvard University, Cambridge, 1974.

[33] D. Whitley and C. Bogart. The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms. In *IEEE International Joint Conference on Neural Networks*, volume I, pages 134–127, 1990.